Date: April 2008

Software & Systems Process Engineering Meta-Model Specification

Version 2.0

OMG Document Number: formal/2008-04-01
Standard document URL: http://www.omg.org/spec/SPEM/2.0/PDF
Associated Files*: http://www.omg.org/spec/SPEM/20070801

http://www.omg.org/spec/SPEM/20070801/SPEM2-Process-Behavior-Content.merged.cmof
http://www.omg.org/spec/SPEM/20070801/Infrastructure.cmof
http://www.omg.org/spec/SPEM/20070801/LM.cmof
http://www.omg.org/spec/SPEM/20070801/SoftwareProcessEngineeringMetamodel.cmof
http://www.omg.org/spec/SPEM/20070801/SPEM2-Method-Content.cmof
http://www.omg.org/spec/SPEM/20070801/SPEM2-Process-Behavior-Content.cmof
http://www.omg.org/spec/SPEM/20070801/SPEM2.cmof
http://www.omg.org/spec/SPEM/20070801/SPEM2.merged.cmof
http://www.omg.org/spec/SPEM/20070801/SPEM2-Method-Content.merged.cmof
http://www.omg.org/spec/SPEM/20070801/SPEM 2.0 Base Plugin Profile.xmi
http://www.omg.org/spec/SPEM/20070801/SPEM 2.0 UML2 Profile.xmi

---

* original .zip files: ptc/07-08-08, ptc/07-08-09

Copyright © 2004-2007, Adaptive Ltd.
Copyright © 2004-2007, Fujitsu
Copyright © 2004-2007, Fundacion European Software Institute
Copyright © 2004-2007, International Business Machines Corporations
Copyright © 1997-2008, Object Management Group
Copyright © 2004-2007, Softeam

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.

IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™ , Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™ , MOF™, OMG Interface Definition Language (IDL)™, and OMG Systems Modeling Language (OMG SysML)™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers, and sellers of computer software to use certification marks, trademarks, or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface xi

1 Scope 1

2 Conformance 1

2.1 Design Principles and Overall Packaging of the SPEM 2.0 Meta-Model 1

2.2 SPEM 2.0 Meta-Model Architecture Overview 2

2.3 Compliance Point "SPEM Complete" 4

2.4 Compliance Point "SPEM Process with Behavior and Content" 5

2.5 Compliance Point "SPEM Method Content" 6

2.6 Additional SPEM 2.0 Implementation Scenarios 7

3 Normative References 7

4 Terms and Definitions 8

5 Symbols 8

6 Additional Information 8

6.1 Background and Rationale 8

6.2 General Introduction to SPEM 2.0 9

6.3 Key New Capabilities of SPEM 2.0 11

6.3.1 Clear separation of method content definitions from the development process application of method content 12

6.3.2 Consistent maintenance of many alternative development processes 15

6.3.3 Many different lifecycle models 16

6.3.4 Flexible process variability and extensibility plug-in mechanism 17

6.3.5 Reusable process patterns of best practices for rapid process assembly 18

6.3.6 Replaceable and reusable Process Components realizing the principles of encapsulation 19

6.4 Specification Formalism 19

6.5 Statement of Proof of Concept and Commercial Availability 22

6.6 Changes to Adopted OMG Specifications 22

6.7 How to Read this Specification 22

6.8 Acknowledgements 23

7 Using SPEM 2.0 as a UML 2 Superstructure Profile 25
7.1 SPEM 2.0 Profile Overview 25
7.2 Describing Work Definitions and Work Breakdown as UML Behavior Models 28
7.3 Describing Work Product Evolution with State Machines 31
7.4 Relating Work Product State to Work Definitions 32
8 Core 35
8.1 Extensible Element 36
8.2 Kind 37
8.3 Parameter Direction Kind 37
8.4 Work Definition 38
8.5 Work Definition Parameter 40
8.6 Work Definition Performer 41
9 Process Structure 43
9.1 Activity 45
9.2 Activity Use Kind 48
9.3 Breakdown Element 54
9.4 Milestone 55
9.5 Process Element 56
9.6 Process Parameter 57
9.7 Process Performer 57
9.8 Process Responsibility Assignment 58
9.9 Role Use 59
9.10 Work Breakdown Element 60
9.11 Work Product Use 62
9.12 Work Product Use Relationship 63
9.13 Work Sequence 66
9.14 Work Sequence Kind 68
10 Process Behavior 69
10.1 Activity_ext 70
10.2 Control Flow_ext 70
10.3 External Reference 70
10.4 State_ext 71
10.5 Transition_ext 71

10.6 Work Definition Parameter 71
11 Managed Content 73
11.1 Category 74
11.2 Content Description 76
11.3 Describable Element 77
11.4 Guidance 78
11.5 Metric 78
11.6 Section 79
12 Method Content 81
12.1 Default Responsibility Assignment 83
12.2 Default Task Definition Parameter 84
12.3 Default Task Definition Performer 85
12.4 Method Content Element 85
12.5 Optionality Kind 86
12.6 Qualification 86
12.7 Role Definition 87
12.8 Step 88
12.9 Task Definition 89
12.10 Tool Definition 91
12.11 Work Product Definition 92
12.12 Work Product Definition Relationship 93
13 Process with Methods 95
13.1 Activity 97
13.2 Breakdown Element 99
13.3 Composite Role 100
13.4 Method Content Kind 101
13.5 Method Content Package 102
13.6 Method Content Packageable Element 103
13.7 Method Content Use 104
13.8 Planning Data 106
13.9 Process Kind 107
13.10 Process Package 108
13.11 Process Packageable Element 109

13.12 Process Performer 109
13.13 Role Use 110
13.14 Task Use 111
13.15 Team Profile 113
13.16 Work Product Use 116
14 Method Plugin 117
14.1 Activity 119
14.2 Method Configuration 120
14.3 Method Library 122
14.4 Method Library Packageable Element 123
14.5 Method Plugin 124
14.6 Method Plugin Packageable Element 126
14.7 Process Component 127
14.8 Process Component Use 131
14.9 Section 132
14.10 Variability Element 133
14.11 Variability Type 137
14.12 Work Product Port 141
14.13 Work Product Port Connector 142
15 Process Diagrams 143
15.1 Workflow Diagram 143
15.2 Activity Detail Diagram 144
15.3 Work Product Dependency Diagram 145
15.4 Team Profile Diagram 145
15.5 Process Component Diagram 146
16 Enacting SPEM 2.0 Processes 147
16.1 Process Enactment with Project Planning Systems 147
16.2 Process Enactment with a Workflow Engine 149
17 Migrating SPEM 1.1 Models to SPEM 2.0 151
18 The SPEM 2.0 Base Plug-in 155
18.1 Activity Kinds 155
18.1.1 Phase 156
18.1.2 Iteration 156

18.1.3 Process 156
18.1.4 Delivery Process 157
18.1.5 Process Pattern 158
18.1.6 Process Planning Template 160
18.2 Category Kinds 160
18.2.1 Discipline 161
18.2.2 Role Set 161
18.2.3 Domain 161
18.2.4 Tool Category 161
18.3 Guidance Kinds 162
18.3.1 Checklist 163
18.3.2 Concept 163
18.3.3 Estimate (metric kind) 163
18.3.4 Estimation Considerations (metric kind) 163
18.3.5 Estimating Metric (metric kind) 163
18.3.6 Example 163
18.3.7 Guideline 163
18.3.8 Practice 164
18.3.9 Report 164
18.3.10 Reusable Asset 164
18.3.11 Roadmap 164
18.3.12 Supporting Material 164
18.3.13 Template 164
18.3.14 Term Definition 164
18.3.15 Tool Mentor 165
18.3.16 Whitepaper 165
18.4 Work Product Kinds 165
18.4.1 Outcome 165
18.4.2 Deliverable 166
18.4.3 Artifact 166
18.5 Work Product Relationship Kinds 167
Annex A: SPEM 2.0 UML 2 Profile Summary 169
Annex B: Additional Variability Examples 175
Annex C: Case Studies and Examples 187
Index 217

List of Figures

Figure 2.1 Structure of the SPEM 2.0 Meta-Model 2
Figure 2.2 Definition of the "SPEM Complete" compliance point 5
Figure 2.3 Definition of the "SPEM Process with Behavior and Content" compliance point 6
Figure 2.4 Definition of the "SPEM Method Content" compliance point 6

Figure 6.1 SPEM 2.0's conceptual usage framework 10
Figure 6.2 Applying the same Method Content (left) in different Processes (right) and different parts of the breakdown structure of the same Process (right-top) 12
Figure 6.3 Method Content definition versus the application of Method Content in a Process 13
Figure 6.4 An alternative presentation for method content versus process 13
Figure 6.5 Key terminology defined in this specification mapped to Method Content versus Process 14
Figure 6.6 A process with variations: A replacement of an Activity depicted in blue color and suppressed Activities in gray color 15
Figure 6.7 A process with an optional step (Define Owner Models) 15
Figure 6.8 Two processes with different lifecycle models. One common SPEM 2.0 structure to represent any of these lifecycles 16
Figure 6.9 An agile process in Macroscope. It comprises three sub-processes, each of which each follows a different lifecycle model 17
Figure 6.10 Example for a method plug-in extending method content and processes with additional capabilities 17
Figure 6.11 A Process Pattern applied three times to a process; each with individual modifications 18
Figure 6.12 Process Components connected via Work Product Ports 19
Figure 6.13 Model Layers for UML and SPEM 2.0 20
Figure 6.14 Exemplary instantiations of the modeling layers 21

Figure 7.1 UML 2 Class Diagram using SPEM 2.0 Profile with stereotyped classes 26
Figure 7.2 The same SPEM 2.0 class diagram of Figure 7-1 using SPEM 1.1 stereotype icons 26
Figure 7.3 Syntactically correct SPEM 1.1 representation of Figure 7-2 27
Figure 7.4 Task represented as stereotyped UML 2 Activity diagram (left) and model objects needed (right) showing Steps as CallBehavior Actions invoking Activities (step icons with the same name) and performing Roles 29
Figure 7.5 Example for an Activity with Task Use 30
Figure 7.6 SPEM 2.0 Activity Workflow represented as UML Activity diagram 31
Figure 7.7 Examples for work product state machines (left) and how states can be used in object flow diagrams to qualify the inputs and outputs of activities 32
Figure 7.8 State transition annotated with mapping work definitions 33
Figure 7.9 Another view of how state transitions relate to work definitions 34

Figure 8.1 The SPEM 2.0 UML 2 Profile stereotypes defined in the Core package 35
Figure 8.2 Extensible Elements can have a Kind 36
Figure 8.3 Work Definition and its related elements 38

Figure 9.1 The SPEM 2.0 UML 2 Profile stereotypes defined in the Process Structure package 43
Figure 9.2 Overview of the main classes and associations of Process Structure package (this diagram does not include all generalizations, see Figure 9-11 for the complete taxonomy) 44
Figure 9.3 Process Structure example: Multi-phased light-weight activity breakdown defining work product use responsibilities 45

Figure 9.4 The Work Definition Activity in relationship to its work performers and input/outputs 46
Figure 9.5 Activity Use and Activity Use Kinds 48
Figure 9.6 Example of an activity reusing another activity 50
Figure 9.7 Using Extends to dynamically bind a Process Pattern to a Delivery Process 50
Figure 9.8 Representation of the Delivery Process from Figure 9.7 before interpretation of the Extends relationship 51
Figure 9.9 Modeling Local Contribution and Local Replacement 52
Figure 9.10 The resulting Delivery Process after interpreting the Variability Specializations 53
Figure 9.11 A breakdown structure is defined by Activities nesting Breakdown Elements 54
Figure 9.12 Taxonomy of classes defined in Process Structure package 56
Figure 9.13 Work Product composition example 64
Figure 9.14 Simple Work Product Dependency Diagram drawing impacts relations as arrows 64
Figure 9.15 Detailed UML 2-based Work Product Dependency diagram showing States as well Deliverable aggregation (Software Requirements Specification) and Work Product composition (Use Case Model) 65
Figure 9.16 Two views on the same process showing a Work Breakdown Structure as well as Work Product Breakdown Structure 66

Figure 10.1 Traceability links from key behavior model abstraction to key Process Elements 69

Figure 11.1 The SPEM 2.0 UML 2 Profile stereotypes defined in the Managed Content package 73
Figure 11.2 The Describable Element parts and subclasses 74
Figure 11.3 Example for categories and catgorized content 75

Figure 12.1 The SPEM 2.0 UML 2 Profile stereotypes defined in the Method Content package 81
Figure 12.2 Taxonomy of Core Describable Elements 82
Figure 12.3 Key Method Content Elements and their relationships 83
Figure 12.4 Example for a UML2 diagram with role stereotypes and artifact responsibility 88
Figure 12.5 Steps represent Sections of Tasks 88
Figure 12.6 Example Task with related content elements represented using the UML 2 SPEM 2.0 Profile 91
Figure 12.7 Work Product taxonomy 92

Figure 13.1 The SPEM 2.0 UML 2 Profile stereotypes defined in the Process with Methods package 95
Figure 13.2 Taxonomy of the meta-model classes defined in the Process with Methods meta-model package 96
Figure 13.3 Taxonomy and key relationships of Breakdown 97
Figure 13.4 Example for an Activity ‘Define the System’ with associations 98
Figure 13.5 Refined Method Content and Process Kinds 101
Figure 13.6 ContentPackage redefines its inherited owned Members association to only allow Method Content Elements 102
Figure 13.7 Example for a Content Package with its Content Elements 103
Figure 13.8 Relationships of concrete Method Content Use classes 104
Figure 13.9 Examples for Method Content Uses (right) referencing Method Content (left) 105
Figure 13.10 Method Content Uses in a Work Breakdown Structure 106
Figure 13.11 Process Packages contains Process specific elements 108
Figure 13.12 A Team Profile consists of Role Uses and/or Composite Roles 110
Figure 13.13 Selecting Steps for a Task Use 111
Figure 13.14 Example of a Team Profile with nested Team Profiles and Role Uses 114
Figure 13.15 Example for Team Profiles used in a Team Breakdown Structure 115

Figure 14.1 The SPEM 2.0 UML 2 Profile stereotypes defined in the Method Plugin package 117

Figure 14.2 Taxonomy of the classes defined in Method Plugin meta-model package 118
Figure 14.3 Definition of Method Library and Method Configurations 120
Figure 14.4 Illustration of a Method Configuration: all colored packages a part of a Configuration 121
Figure 14.5 A Method Library is a container for Plugins and Configurations 122
Figure 14.6 Method Plugins are containers for Method Packages 124
Figure 14.7 Example Method Plugin with sample contents and its relationships 125
Figure 14.8 Specification of Process Components and their relationships 127
Figure 14.9 Graphical Representation of a Process Component 128
Figure 14.10 Three Process Components to be assembled 129
Figure 14.11 Assembling Process Components 129
Figure 14.12 Introduction of an intermediary Work Product in the assembly process 130
Figure 14.13 Introduction of a missing input Work Product 130
Figure 14.14 Defining correspondences between different model elements 131
Figure 14.15 Overview to Variability for Content Elements and Processes 133
Figure 14.16 Example for variability using Contribution on Roles, Task, and Work Products 135
Figure 14.17 Result after interpreting Contribution in Figure 14-15 136

Figure 16.1 Mapping between Activity Diagram, Process, and Project Plan 148
Figure 16.2 A second Process to Plan mapping example 149

Figure 18.1 The Activity Kinds Stereotypes 155
Figure 18.2 Example for Process Pattern defining Requirements Management work 159
Figure 18.3 Relationships among Processes 159
Figure 18.4 The Category Kinds Stereotypes 160
Figure 18.5 The Guidance Kinds Stereotypes 162
Figure 18.6 The Work Product Kinds Stereotypes 165
Figure 18.7 The Work Product Relationship Kinds 167

Figure C.1 Conceptual mapping of MSF Agile to SPEM 2.0 concepts 191
Figure C.2 Side-by-side comparison of a Role presentation and associated concepts in MF Agile (left) and modeled with SPEM 2.0 concepts (right) 192
Figure C.3 Mapping of an MSF Activity to a SPEM 2.0 Task 193
Figure C.4 Mapping of MSF Agile Workstreams to SPEM 2.0 Process Patterns 194
Figure C.5 Modeling MSF Agile process Views as SPEM 2.0 Process Patterns 195
Figure C.6 Eclipse Process Framework Ecosystem accommodating content from many different sources 196
Figure C.7 Overview to OpenUp/Basic showing its key roles, disciplines that categorize tasks, and work products as well as the overall lifecycle model 197
Figure C.8 Typical OpenUP process presented as a breakdown structure as well as activity diagram 198
Figure C.9 'Platform Export' Role Overview 199
Figure C.10 Process Patterns realizing MDA 199
Figure C.11 'Outline PSM Model' Capability 200
Figure C.12 Some of the 17 IT processes (left) represented by ITUP as Process Patterns and so-called process scenario that also have been assembled as Process Patterns using elements from the IT processes as building blocks 201
Figure C.13 Asset Management IT Process represented as a SPEM 2.0 Process Pattern that uses ITUP's own proprietary behavior modeling approach to present the process' workflow 202
Figure C.14 A process scenario illustrating how IT processes work together to perform common IT functions 203
Figure C.15 Overview to the SPI – PM Portal PMBOK representation 204
Figure C.16 Overview to the Project Manager role 205

Figure C.17 PMBOK Process Group modeled as an activity diagram 206
Figure C.18 A role defined by the SOA Governance Method defining its key tasks, responsible work products, and work products the role modifies when performing the tasks 207
Figure C.19 SOA lifecycle for introducing or improving SOA Governance in an organization 208
Figure C.20 Documented CMM Practices are linked to enactable process models (Process Patterns) and artifacts used and produced in the practice 209
Figure C.21 List of Process Patterns that model so-called OPAL procedures that can be used as reusable building blocks to assemble larger end-to-end processes 210
Figure C.22 A workflow in the ASIC method (top) defining control flow among task uses as well as list of the same task uses grouped by performing role 211
Figure C.23 General ISPW-6 example breakdown structure 212
Figure C.24 Flow of activities in the ISPW-6 example 213
Figure C.25 Role Definition for Design Engineer in the ISPW-6 Example 214
Figure C.26 The Money Lover website 215
Figure C.27 Investment club lifecycle model defining three phases and the detailed workflow for the first phase 216

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A - Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

The purpose of this document is to provide a comprehensive definition of the Software & Systems Process Engineering Meta-Model 2.0 (SPEM 2.0). It serves as a guide for understanding the semantics of this meta-model as well as its direct application for all method and process modeling activities. The goal is to accommodate a large range of processes, and not to exclude them by having too many features or constraints. The metamodel uses UML as a notation and takes an object-oriented approach.

2 Conformance

This specification defines three compliance points for SPEM 2.0. Implementations are encouraged to conform to one of these compliance points if their goal is to ensure successful data exchange with other compliance point implementers. In addition to these compliance points, the specification provides the freedom to implementers to choose any combination of meta-model packages and package merges that they wish to implement (see Section 2.6 for a discussion). However, if data exchange is a major goal for an implementer, we encourage implementing one of these compliance points.

SPEM 2.0 is defined as a meta-model as well as a UML 2 Profile. If the SPEM 2.0 UML 2 Profile is used by the implementer, then the same compliance points apply in the sense that stereotypes get introduced in the same specification chapters as their respective meta-model classes. Hence a compliance point includes all stereotypes defined in the chapters that have been listed for inclusion in the definition of each compliance point below. However, only one profile including all stereotypes is provided as part of this specification (OMG document ad/06-11-05).

The specification provides individual XMI schemata for all three compliance points (OMG document ad/06-11-04). The compliance points listed here are defined by the inclusion and merge of specific meta-model packages. The following Section 2.1 and Section 2.2 provide an overview to all meta-model packages available in SPEM 2.0. Section 2.3 to Section 2.5 will then define the three compliance points. Finally, Section 2.6 discusses other non-compliant implantation scenarios that might be useful for specific audiences of SPEM 2.0.

2.1 Design Principles and Overall Packaging of the SPEM 2.0 Meta-Model

The SPEM 2.0 Meta-Model is a MOF-based [MOF 2] model that reuses other OMG specifications. SPEM 2.0 reuses the UML 2 Infrastructure library [UML 2 Infrastructure] wherever possible. Key concepts and structures such as Classifier and Package have been directly reused by sub-classing SPEM 2.0 classes from these. A SPEM 2.0 implementer should also utilize the UML 2 diagram interchange [UML 2 Diagrams] for the presentation of various diagram types as presented in examples throughout this document and summarized in Chapter 15.

Within the package named “SPEM” you will find the actual architecture of the SPEM meta-model as depicted in Figure 2.1. SPEM 2.0 applies the UML 2 Infrastructure’s package merge mechanism to gradually build up the meta-model, providing optional meta-model packages or modular units as building blocks for a specification implementer. In other words an implementer or adopter can choose to utilize different levels of capabilities, number of concepts, and levels of formalism for expressing their processes by using or realizing only certain packages of this architecture. We define the three most typical selections as the SPEM 2.0 Compliance Points, but also discuss other possibilities of combining the meta-model packages to address more specific process modeling needs.

2.2 SPEM 2.0 Meta-Model Architecture Overview

SPEM 2.0 is used to define software and systems development processes and their components. The scope of SPEM is purposely limited to the minimal elements necessary to define any software and systems development process, without adding specific features for particular development domains or disciplines (e.g., project management). The goal is to accommodate a large range of development methods and processes of different styles, cultural backgrounds, levels of formalism, lifecycle models, and communities. However, the focus of SPEM is development projects. SPEM 2.0 does not aim to be a generic process modeling language, nor does it even provide its own behavior modeling concepts. SPEM 2.0 rather defines the ability for the implementer to choose the generic behavior modeling approach that best fits their needs. It also provides specific structures to enhance such generic behavior models that are characteristic for describing development processes. In other words, SPEM 2.0 focuses on providing the additional information structures that you need for processes modeled with UML 2.0 Activities or BPMN/BPDM to describe an actual development process.

The SPEM 2.0 meta-model is structured into seven main meta-model packages as depicted in Figure 2.1. The structure divides the model into logical units. Each unit extends the units it depends upon, providing additional structures and capabilities to the elements defined below. Overall, the UML 2 package merge mechanism applied to the packages realizes a gradual extension of the capabilities modeled unit by unit. As a result, units defined on a lower layer can be realized by a SPEM 2.0 subset-implementation without the higher level units. In many cases, meta-model classes are introduced in a lower level package as simply as possible, and are then extended in higher level units via the package merge mechanism with additional properties and relationships to realize more complex process modeling requirements.

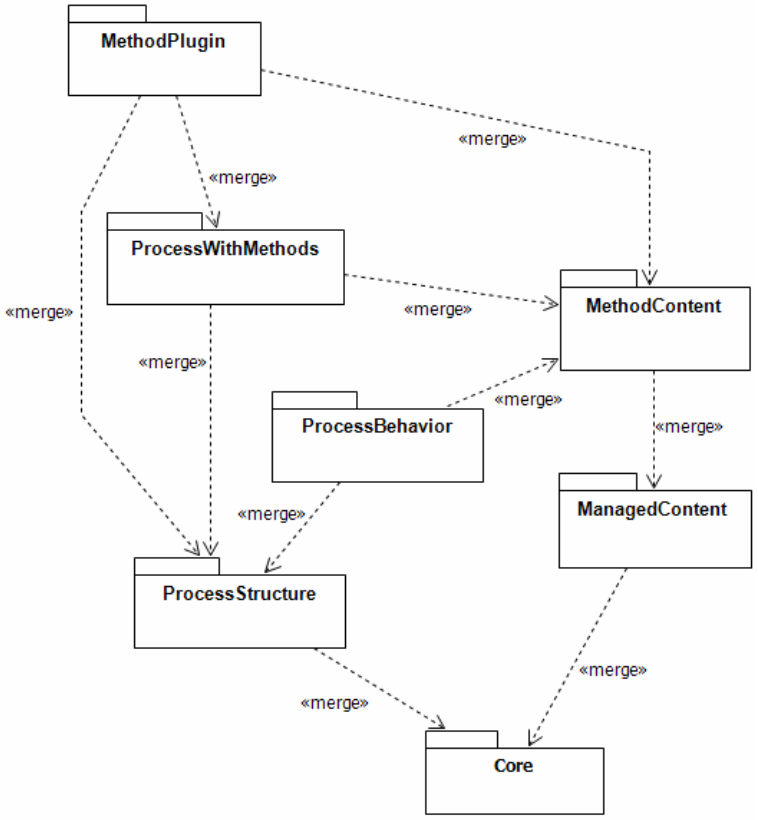


Figure 2.1 - Structure of the SPEM 2.0 Meta-Model

The packages depicted in Figure 2.1 provide the following capabilities:

- **Core**: The Core meta-model package contains those meta-model classes and abstractions that build the base for classes in all other meta-model packages. In other words, all the common classes among all compliance levels defining the core of SPEM 2.0 have been placed here. Core mainly defines classes for two SPEM 2.0 capabilities: (1) The ability for a SPEM 2.0 user to create user-defined qualifications for a SPEM 2.0 classes allowing users to distinguish different 'kinds' of SPEM 2.0 class instances. (2) A set of abstract classes to define work expressed as SPEM 2.0 processes. All SPEM 2.0 classes that derive from these classes are intended to map to behavior classes of behavior models (e.g., can be assigned as stereotypes to UML 2.0 Activities or linked to behaviored classifiers).

- **Process Structure**: This package defines the base for all process models. It supports the creation of simple and flexible process models. Its core data structure is a breakdown or decomposition of nested Activities that maintain lists of references to performing Role classes as well as input and output Work Product classes for each Activity. In addition, it provides mechanisms for process reuse such as the dynamic binding of process patterns that allow users to assemble processes with sets of dynamically linked Activities. These structures are used to represent high-level and basic processes that are not textually documented. The structures are ideal for the ad-hoc assembly of processes, especially the representation of agile processes and self-organizing team approaches.

- **Process Behavior**: The concepts of the Process Structure package represent a process as a static breakdown structure, allowing nesting of activities and defining predecessor dependencies among them. The Process Behavior meta-model package allows extending these structures with behavioral models. However, it does not define its own behavior modeling approach, but rather provides ‘links’ to existing externally-defined behavior models, enabling reuse of these approaches from other OMG or third party specifications. For example, a process defined with the Process Structure concepts can be linked to UML 2 Activity diagrams that represent the behavior of such process; or a Work Product Definition from the Method Content package can be linked to a state machine model that represents its typical lifecycle. Chapter 7 shows examples for such process models that utilize the UML 2 profile defined in this specification for a consistent presentation for such UML 2 models in addition to the ‘links’ defined in this Process Behavior meta-model package.

- **Managed Content**: Development processes are in many cases not only represented as models, but documented and managed as natural language descriptions. For many software development approaches and methods, human-consumable documentation providing understandable guidance for best development practices is more important than precise models. In other words, the practicality of techniques and methods expressed with these practices is in many cases perceived to provide higher value than strict obedience to a formally defined process. The reasons for this are that many development approaches see software development as a creative process that requires constant re-evaluation and adoption rather than a strict sequence of activities. For example, for modern agile development teams, best practices of software development are communicated through mentoring and short practice descriptions in white paper format, rather than formally defined models. They assume that certain values and a development culture (in other words the social engineering required for agile development) cannot be formalized with models, but can only be captured in natural language documentation. The Managed Content meta-model package introduced concepts for managing the textual content of such descriptions. These concepts can either be used standalone or in combination with process structure concepts. For example, a SPEM 2.0 based process could be solely comprised of a set of instances of the guidance meta-class defining development best practices in whitepaper format. It could also be comprised of a combination of these guidance elements with a process structure using relationships defined in the Managed Content meta-model package that allows associating guidance elements with process structure elements.

- **Method Content**: The Method Content meta-model package provides the concepts for SPEM 2.0 users and organizations to build up a development knowledge base that is independent of any specific processes and development projects. It adds concepts for defining lifecycle and process-independent reusable method content elements that provide a base of documented knowledge of software development methods, techniques, and concrete realizations of best practices. Method Content comprises of textual step-by-step explanations, describing how specific fine-granular development goals are achieved by which roles with which resources and results, independent of the placement of these

steps within a specific development lifecycle. Processes would reuse these method content elements and relate them into partially-ordered sequences that are customized to specific types of projects (see 6.3.1, 'Clear separation of method content definitions from the development process application of method content' for more details). A SPEM 2.0 user can define Method Content as general guidance and build up a knowledge base of development methods without ever creating a process, but adding a little more structure for her content as provided by the generic meta-classes defined in the Managed Content package. These structures selected for the Method Content package have been derived from best practices in the industry. Development processes can be based on reusable method content (as defined in the Process with Methods meta-model package), but they can also be independent of method content (by just using the Process Structure meta-model package), thus defining ad-hoc processes that are not based on reusable methods.

- **Process With Methods**: The Process with Methods meta-model package defines new and redefines existing structures for integrating processes defined with Process Structure meta-model package concepts with instances of Method Content meta-model package concepts. Whereas Method Content defines fundamental methods and techniques for software development, processes place these methods and techniques into the context of a lifecycle model comprising, for example, of phases and milestones. When applying method content, such as Tasks, Roles, and Work Products to specific parts of the process, reference classes (referred to as Method Content Use in this specification) are created that can store individual changes to their referenced Method Content classes. These changes express how and which parts of the method will be applied in that particular point in the process.
- **Method Plugin**: The Method Plug-in meta-model package introduces concepts for 'designing' and managing maintainable, large scale, reusable, and configurable libraries or repositories of method content and processes. The concepts introduced in this package allow arranging different parts of such a library based on different layers of concern similar to layered software architectures. With concepts such as Method Plug-in, Process Component, and Variability, one can define processes that are granularly extended with more and more capabilities. Users can then select the process capabilities they are interested in by defining so-called method configurations. Only those selected capabilities will then be surfaced within these configurations to the end-user, allowing process authors to define consistent and maintainable processes for different audiences that are configurable for specific end-user needs.

2.3 Compliance Point "SPEM Complete"

Audience: Large scale process and method library tool providers

The compliance point "SPEM Complete" comprises all seven SPEM meta-model packages described in 2.2, 'SPEM 2.0 Meta-Model Architecture Overview.' Figure 2.2 shows that the compliance point creates a namespace called "SPEM2-Complete" that merges the LM compliance level from the UML 2 Infrastructure Library, UML 2 Profiles, as well as the SPEM 2.0 meta-model packages Method Plugin and Process Behavior, which transitively merge in all other SPEM 2.0 meta-model packages (see Figure 2.1). These seven meta-model packages are described in Chapters 8 to 14 of this specification.

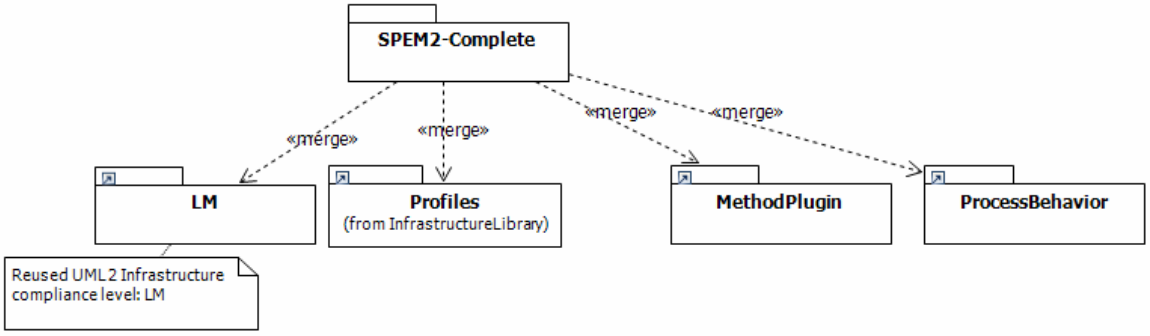


Figure 2.2 - Definition of the "SPEM Complete" compliance point

This compliance point is recommended for implementers who need all capabilities defined in this meta-model. It is aimed at CASE tool providers that want to provide support for large scale libraries of textual method content and reusable process models. The focus is on managing many processes for complex multi-tiered organizations that manage interrelated processes. It is also the only compliance point that merges the Profiles package from the UML 2 Infrastructure, which provides a more complete extensibility mechanism than the light extensibility mechanisms provided in SPEM 2.0 itself. Whereas SPEM 2.0 provides the ability to define instances of a Kind class that allow associating special semantics to meta class instances, UML Profiles allow in addition to that creating and managing stereotype application instances that can store user-defined property values defined for the stereotype with the stereotype instance. SPEM Complete implementers shall either provide both extensibility mechanisms or a mapping of UML Profile stereotypes to SPEM 2 Kinds. The other compliance levels defined in the specification do not require realizing Profiles because the light-weight extensibility mechanisms of SPEM should be sufficient for the audiences listed, but implementers have the option to do so for these levels as well.

The CASE tools mentioned in 6.5, 'Statement of Proof of Concept and Commercial Availability' implement this compliance point.

2.4 Compliance Point "SPEM Process with Behavior and Content"

Audience: SPEM 1.x backwards-compatible and modeling focused tool providers

The compliance point "SPEM Process with Behavior and Content" comprises four of the SPEM meta-model packages described in 2.2, 'SPEM 2.0 Meta-Model Architecture Overview.' Figure 2.3 shows that the compliance point creates a namespace called "SPEM2-Process-Behavior-Content" that merges the LM compliance level from the UML 2 Infrastructure Library as well as the SPEM 2.0 meta-model packages Managed Content (Chapter 11) and Process Behavior (Chapter 10), which transitively merge in (see Figure 2.1) Process Structure (Chapter 9) and Core (Chapter 8).

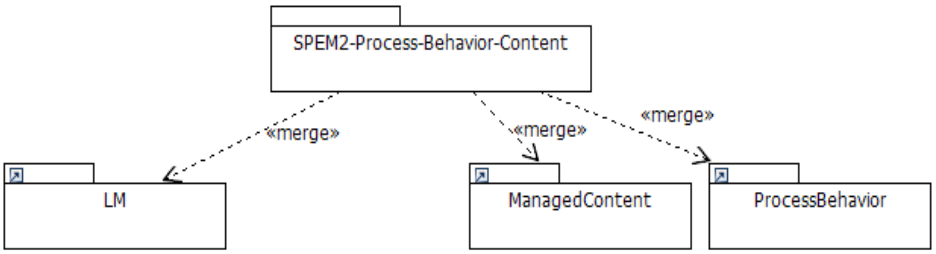


Figure 2.3 - Definition of the "SPEM Process with Behavior and Content" compliance point

This compliance point is recommended for implementers who want to focus on the modeling aspects of SPEM. It aims at audiences that focus on one process at a time and primarily work on process model representations such as work breakdown structures or workflow diagrams. Although this compliance point provides the ability to document the processes textually using the Managed Content concepts, its audience does not require the capabilities of Method Content or Method Plug-in to provide reusable method descriptions nor do they require variability concepts that allow different configurations of variable models and text.

This compliance point is also very close to capturing the capabilities of the predecessor SPEM 1.x specification and does not comprise the additional capabilities introduced in SPEM 2.0 for development knowledge bases of method content, scaling, and variability.

2.5 Compliance Point "SPEM Method Content"

Audience: Method documenter and book authors, organizational knowledge base provider

The compliance point “SPEM Method” comprises three of the SPEM meta-model packages described in Section 2.2. Figure 2.4 shows that the compliance point creates a namespace called “SPEM2-Method-Content” that merges the LM compliance level from the UML 2 Infrastructure Library as well as the SPEM 2.0 meta-model package Method Content (Chapter 12), which transitively merges in (see Figure 2.1) Managed Content (Chapter 11) and Core (Chapter 8).

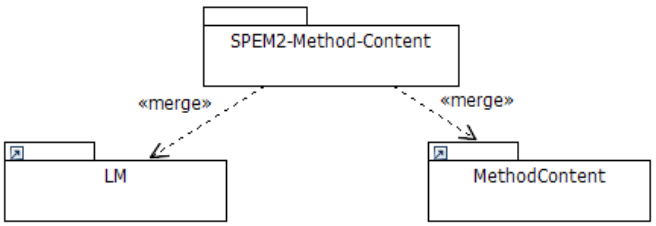


Figure 2.4 - Definition of the "SPEM Method Content" compliance point

This compliance point is recommended for implementers who primarily focus on managing the documentation of descriptions of development methods, techniques, and best practices. Implementations of this level can be used to create development knowledge bases as well as serve as a structure for Wiki-based and other collaborative information systems.

The typical audience of this level in many cases does not need or want formal process models as a representation for their content. They provide descriptions of their methods with a minimal set of concepts (which could be even a sub-set of concepts available in this compliance point) such as Work Product, Task, and Role Definitions, or even less formal Guidance concepts such as guidelines or white papers that adequately represent their agile view of communicating development knowledge.

This compliance point is also ideal for book and technical report authors that could augment their publication of development methods, technique, or best practice with a semi-formal SPEM Method Content-based documentation providing a reusable and interchangeable format of their content using these SPEM concepts.

2.6 Additional SPEM 2.0 Implementation Scenarios

Section 2.2 to Section 2.5 specify three compliance points defining the most typical levels for implementers to choose from. However, as the meta-model packages gradually provide more capabilities along their merge dependencies, many other combinations of meta-model packages are possible and might provide value for other purposes. For example, one could choose to implement the following combinations below: (All of these combinations would need to include the Core package. It has therefore not been explicitly listed below.)

- *Process Structure and Process Behavior*: An implementer might choose to provide a pure modeling solution for SPEM 2.0 or a solution in which the UML 2 Infrastructure ability of attaching comments to each model element is sufficient. The implementer would therefore not realize the Managed Content package that provides additional documentation as well as content categorization capabilities.
- *Process Structure and Managed Content*: This combination does not include Process Behavior. If an implementer wants to solely focus on breakdown structure representations of processes, but wants to document and publish the process models with textual documentation, the implementer can choose to implement these two meta-model packages only. As a result, every Process Element (Section 9.1) of a process breakdown structure can be documented with textual Content Descriptions (Section 11.1). Furthermore, textual Guidance elements (Section 11.4) such as checklists, templates, examples, guidelines, and Metrics (Section 11.5) can be created and systematically linked to these process elements.
- *Process Structure, Process Behavior, and Method Plugin*: If an implementer's focus is on providing libraries of reusable process models without documenting these process models, the implementer could choose to combine the Process Structure and Behavior meta-model packages with the Method Plug-in package. This will enable the implementer to define libraries (Section 14.3) of reusable Processes (Section 9.1) as well as dynamic process extensions using variability (Section 14.10) and to organize these processes and extensions in configurable plug-ins (Section 14.1 and Section 14.2).
- *Process with Methods, Process Structure, Method Content, Managed Content*: This combination would be of interest for an implementer who might want to provide a small-scale realization of SPEM 2.0. The implementer might be interested in fully utilizing the separation of method content from processes, but does not need the capabilities of variability and managing Method Libraries, Method Plugins, and Configurations defined in the Method Plugin meta-model package.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- [MOF 2] OMG, Meta-Object Facility Version 2, www.omg.org/mof.
- [UML 2 Infrastructure] OMG, UML 2 Infrastructure Specification, www.omg.org/uml.
- [UML 2 Diagrams] OMG, Unified Modeling Language Version 2 Diagram Interchange, www.omg.org/uml.

4 Terms and Definitions

There are no formal definitions in this specification that are taken from other documents.

5 Symbols

There are no symbols defined in this specification.

6 Additional Information

6.1 Background and Rationale

This is the third specification defined for the Software and Systems Process Engineering Meta-Model. The specification of SPEM 1.0 was released in 2002 (final FTF report in May 2002). SPEM 1.1 incorporated minor updates that were formally released in 2005 (final RTF report in March 2004).

SPEM 1.x was defined as both a standalone meta-model built upon UML 1.4, and a UML Profile, and was accompanied by an XML DTD. The meta-model used UML 1.4 as a notation and took an object-oriented approach to representing behavior of developers as their operations. SPEM 1.x saw low uptake. Since its issuance, few implementations have been released and it has not been recognized by industry analysts who also failed to acknowledge its relevance to the methodology and process tools market. There have been a number of low-profile or casual adopters of the specification as well as few commercial implementations. It is suspected that ease of adoption has been an issue, and some of the SPEM 1.x semantics were ambiguous and hard to understand by adopters and hence not used in their practices.

Now that major revisions to the underlying UML standard have been developed in UML 2, there are major benefits to be reaped in SPEM. UML 2 includes new features such as greatly improved modeling techniques, and graphic interchange capability, which are of obvious use in process modeling. Furthermore, UML 2 is organized in a more modular fashion so as to allow related specifications to reuse only the required parts of the UML meta-model. The ability to leverage these features, as well as the ability to work with UML 2 tools are powerful enhancements to SPEM 2.0. In addition, there was specific feedback from implementers of SPEM 1.x that have been addressed to make SPEM process models easier to enact and automate. This specification addresses the following requirements from the RFP:

- Update SPEM to be compliant with UML 2, taking advantage of the new functionality to improve process modeling techniques and capabilities.
- Define a new SPEM XML Schema, based on MOF 2.0. XML Schemas provide greater richness and control beyond what is available in the SPEM 1.1 XMI DTD.

- Provide guidance on migrating existing process models from SPEM 1.1 to SPEM 2.0.
- Address feedback from first implementers to address identified inconsistencies and concerns regarding the practicality and functional coverage of SPEM 1.1.
- Define extensions to SPEM that will be of use to process automation tools.
- Align SPEM with evolving and emerging standards other than UML; specifically, the Business Process Definition Meta-model and the Business Process Runtime Interfaces submission may be able to be used in conjunction with SPEM to provide greater value to the user community.
- Introduce process meta-model extensions that may be used equally in software development processes and systems engineering processes.

6.2 General Introduction to SPEM 2.0

Throughout the software industry there are a lot of great ideas and knowledge available about how to effectively develop software. Nowadays, development teams need and have access to a wide range of information. Not only do they need to acquire detailed information about specific development technologies such as Java, Java EE, Eclipse, SOA technologies, .NET, as well as various development and tool environments, but they also need to figure out how to organize their work along modern development best practices such as agile, iterative, architecture-centric, risk- and quality-driven software development.

Some problems development organizations face when they leave their developers to find such information for themselves are:

- team members will not have centralized and easy access to the same body of information when they need it, i.e., different developers might rely on different sources and versions of the same information;
- it is difficult to combine and integrate content and development processes that are made available in their own proprietary format, as every book and publication presents method content and process using a different representation and presentation style;
- it is hard to define an organized and systematic development approach that is right-sized to their needs, i.e., addresses their specific culture, standardized practices, and compliance requirements.

The Software and Systems Process Engineering Meta-model (SPEM) is a process engineering meta-model as well as conceptual framework, which can provide the necessary concepts for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes. An implementation of this meta-model would be targeted at process engineers, project leads, project and program managers who are responsible for maintaining and implementing processes for their development organizations or individual projects.

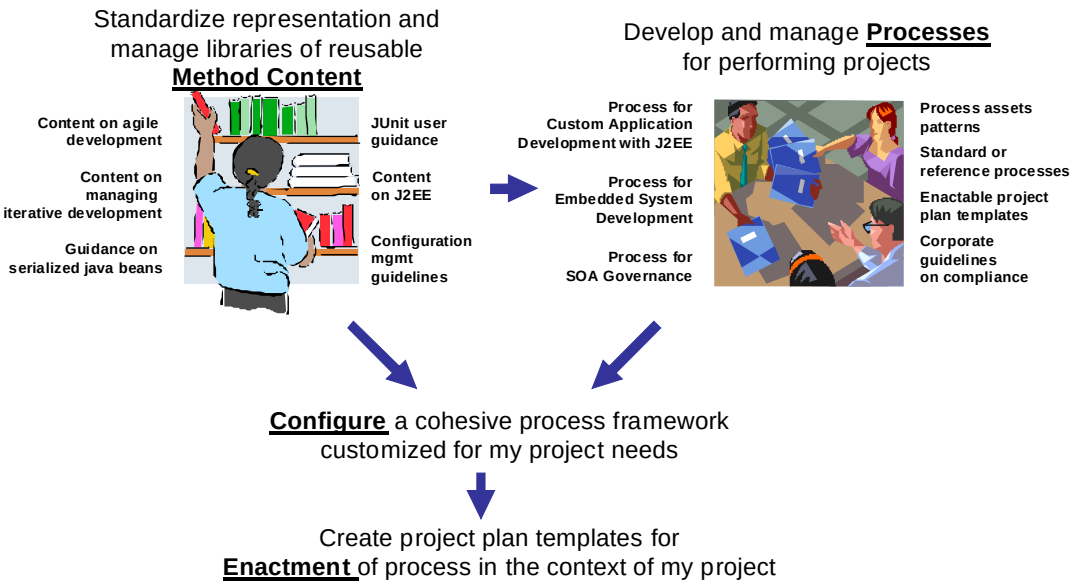


Figure 6.1 - SPEM 2.0's conceptual usage framework

Figure 6.1 presents a conceptual framework for the usage of a SPEM 2.0 implementation:

- **To provide a standardized representation and managed libraries of reusable method content**: Developers need to understand the methods and key practices of software development. They need to be familiar with the basic development tasks, such as how to elicit and manage requirements, how to do analysis and design, how to implement for a design or for a test case, how to test implementations against requirements, how to manage the project scope and change, and so on. They further need to understand the work products such tasks create as well as which skills are required. SPEM 2.0 aims to support development practitioners in setting-up a knowledge base of intellectual capital for software and systems development that would allow them to manage and deploy their content using a standardized format. Such content could be licensed, acquired, and, more importantly, their own homegrown content consisting of, for example, method definitions, whitepapers, guidelines, templates, principles, best practices, internal procedures and regulations, training material, and any other general descriptions of how to develop software. This knowledge base can be used for reference and education and forms the basis for developing processes (see the next bullet point).
- **To support systematic development, management, and growth of development processes**: Development teams need to define how to apply their development methods and best practices throughout a project lifecycle. In other words, they need to define or select a development process. For example, requirements management methods have to be applied in one fashion during the early phases of a project, where the focus is more on elicitation of stakeholder needs and requirements and scoping a vision. The same methods have to be performed in a different fashion during later phases, where the focus is on managing requirements updates and changes and performing impact analysis of these requirements changes. The same requirements methods might also be applied differently if the project develops a new system or maintains an existing system as well as depending on the teams and distribution of the teams. A development process model needs to support expressing these differences. Teams also need a clear understanding of how the different tasks within the methods relate to each other: for example, how the change management method impacts the requirements management method as well as the regression testing method through out the lifecycle. SPEM 2.0 supports the systematic creation of processes based on reusable method content. Lifecycle independent method content can be placed into a process for a specific development lifecycle. Such processes can be represented as workflows and/or breakdown structures. Within these process the reused method content can then be refined for its specific context. SPEM 2.0 also provides the conceptual foundation for process engineers and project managers for

selecting, tailoring, and rapidly assembling processes for their concrete development projects. The vision for SPEM 2.0 is that vendors can sell with their SPEM 2.0 implementation catalogs of pre-defined processes for typical project situations that can be adapted to individual needs. Within these catalogs, the implementation could offer process building blocks or process patterns that represent references processes for specific disciplines, technologies, or development styles. These process patterns could form a toolkit for quickly assembling processes based on project specific needs.

- **To support deployment of just the method content and process needed by defining configurations of processes and method content**: No development project is exactly like another and the same development process is never executed twice. Reference frameworks for development processes such as CMMI define different levels of maturity for processes. Each level entails different characteristics for the process definition as well as enactment in an organization or project for each level. For example, CMMI defines a "managed process" as performed activities that can be recognized as implementations of development practices. Such a process has certain characteristics: it is planned and executed in accordance with policies; it employs skilled people, having adequate resources to produce controlled outputs; it involves relevant stakeholders; it is monitored, controlled, and reviewed; and it is evaluated for adherence to its process description. By contrast, a "defined process" is a managed process that is tailored from the organization's set of standard processes according to the organization's tailoring guidelines. A defined process has a maintained process description and contributes work products, measures, and other process-improvement information to the organizational process assets. The notions of Activity Use, Configurability, and Variability for development processes (as well as method content) in SPEM 2.0 exactly address the needs for defined processes. These concepts provide capabilities for reuse of processes or process patterns, for modeling variability (i.e., processes that comprise of configurable alternative parts) and for tailoring allowing users to define their own extensions, omissions, and variability points on reused standard processes. Hence, the SPEM usage scenario is that organizations can provide libraries of reusable process and method using the capabilities describes in the first two bullet points. Team leads can then select and tailor the method content and processes that they require. They can then describe these selections and customizations with a SPEM Method Configuration, which they can deploy to their teams, only providing the content they really need.
- **To support the enactment of a process for development projects**: A process definition only provides value if it impacts and steers development teams' behavior. Processes as well as guiding method content need to be available in the context of daily work of project managers, technical leads, and developers. They therefore need to be deployed in formats that are ready for enactment with the process enactment systems of the team's choice. Typical enactment systems are project and resource planning systems, work backlog tracking systems, and workflow engines. SPEM 2.0 provides process definition structures that allow process engineers to express how a process shall be enacted within these systems. For example, SPEM 2.0 process definition can include information that indicates that modeled work definitions shall be repeated several times in a project (modeling iterations) or that there could be multiple occurrences of work definitions that can be performed in parallel.

Although, the SPEM 2.0 meta-model has been designed around the support for this framework, many other usage scenarios could be realized as well. For example, Chapter 2 defines different compliance points and discusses different implementation scenarios that might realize a variation of the scenarios depicted in Figure 6.1.

6.3 Key New Capabilities of SPEM 2.0

In addition to addressing the RFP requirements listed above, this specification provides the following new capabilities for process authors.

6.3.1 Clear separation of method content definitions from the development process application of method content

As outlined in Section 6.2, SPEM 2.0 separates reusable method content from its application in processes. Method content provides step-by-step explanations, describing how specific development goals are achieved independent of the placement of these steps within a development lifecycle. Processes take these method content elements and relate them into partially-ordered sequences that are customized to specific types of projects.

For example, a software development project that develops an application from scratch performs development tasks such as “Find Actors and Use Cases,” “Develop Vision,” or “Design Use Case” similarly to a project that extends an existing software system. However, the two projects will perform the Tasks at different points in time with a different emphasis, i.e., they will perform the steps of these tasks differently, assume different inputs, and perhaps apply individual variations and additions.

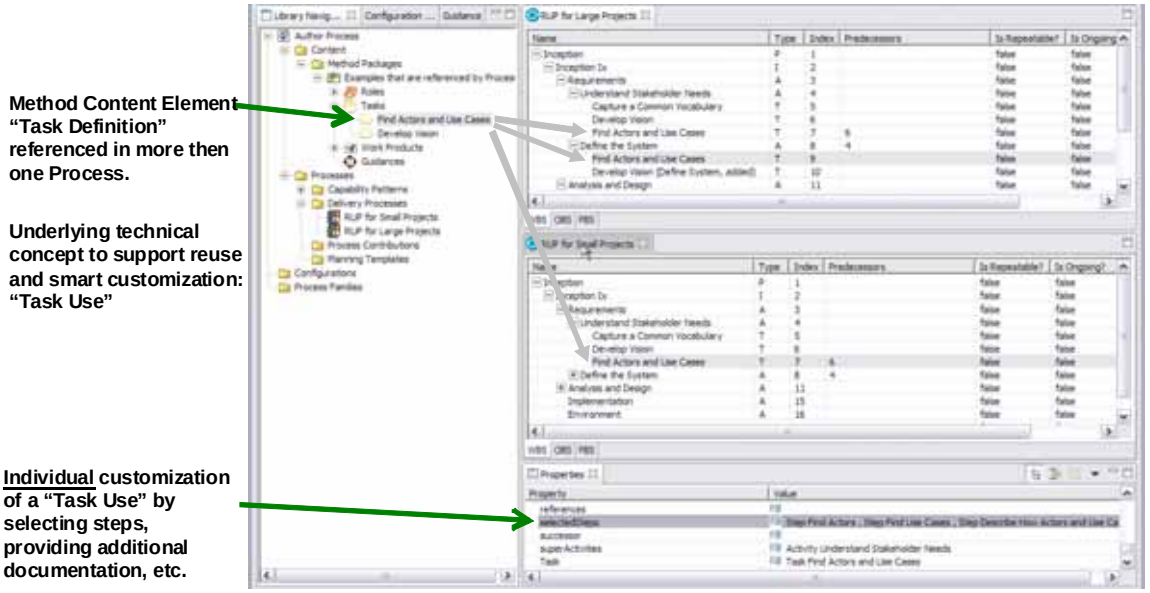


Figure 6.2 - Applying the same Method Content (left) in different Processes (right) and different parts of the breakdown structure of the same Process (right-top)

Figure 6.2 depicts an example from a SPEM 2.0 implementation. It shows that SPEM 2.0 allows each process on the right to reference common method content, such as definitions for roles, tasks, and work products, as well as general guidance from a common method content pool depicted on the right. These references realize traceability for processes to their underlying method content, allowing changes in the methods to be reflected in all processes using it. Moreover, SPEM 2.0 still allows overriding certain method related content within a process as well as defining individual process-specific relationships for each process element (such as work breakdown and new relations to work products and roles).

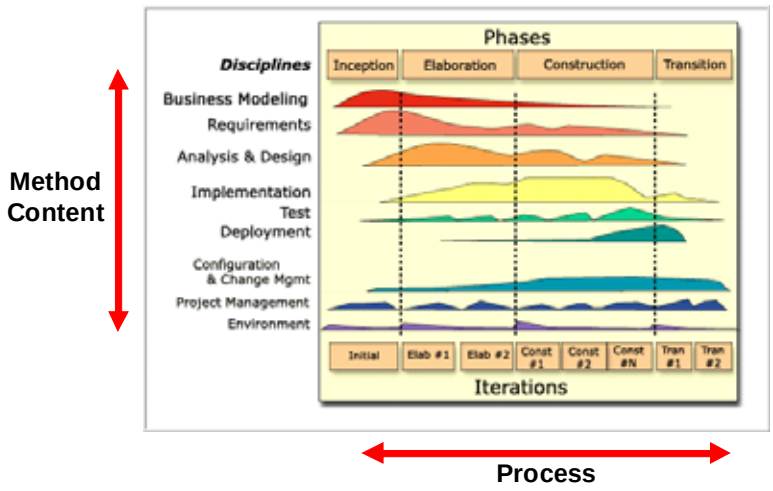


Figure 6.3 - Method Content definition versus the application of Method Content in a Proces

Figure 6.3 (taken from the Unified Process) shows the difference between method content and process by representing them as two different dimensions. Method content describing how development work is being performed is categorized by disciplines in this example. Each discipline is comprised of task definitions (not visible in Figure 6.3) that provide step-by-step descriptions of how specific development goals are achieved. In a process, task definitions will be selected from the method content and placed into workflows as task uses ready for instantiation. Instantiation would allocate resources to perform the work and assign real work products as the inputs and outputs of the tasks. The workload graphs of Figure 6.3 show work effort for each discipline over time (from left to right).

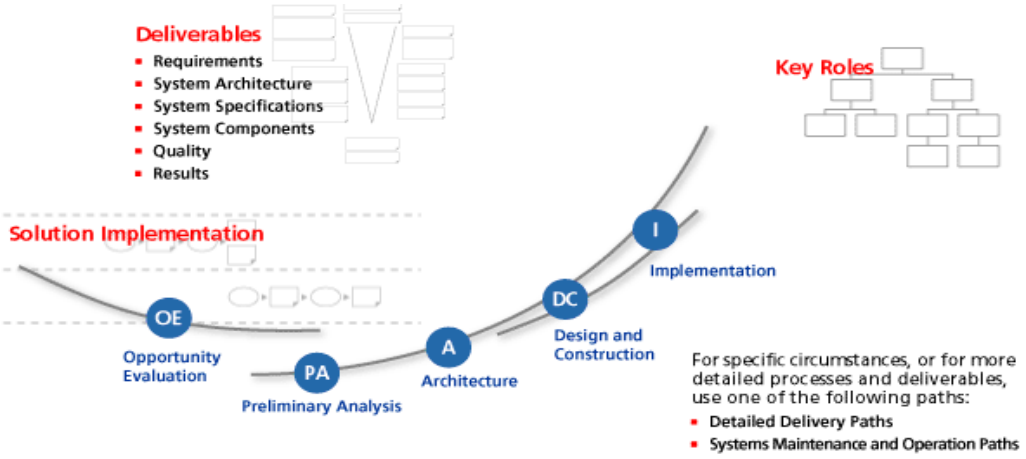


Figure 6.4 - An alternative presentation for method content versus process

Figure 6.4 shows an alternative presentation (taken from Fujitsu Macroscope) for the separation of method content from processes. It shows how common method content (Deliverables and Key Roles) definition and structure are used by a variety of standard processes. A process determines the scope and level of details of the deliverables and orchestrates their production by key roles.

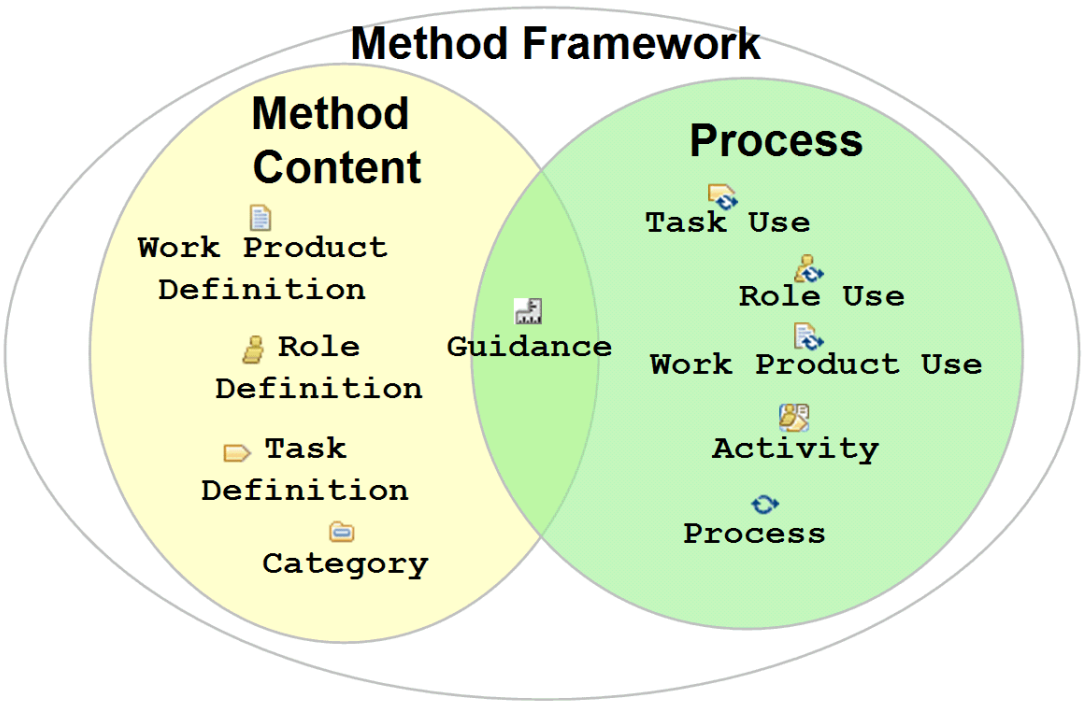
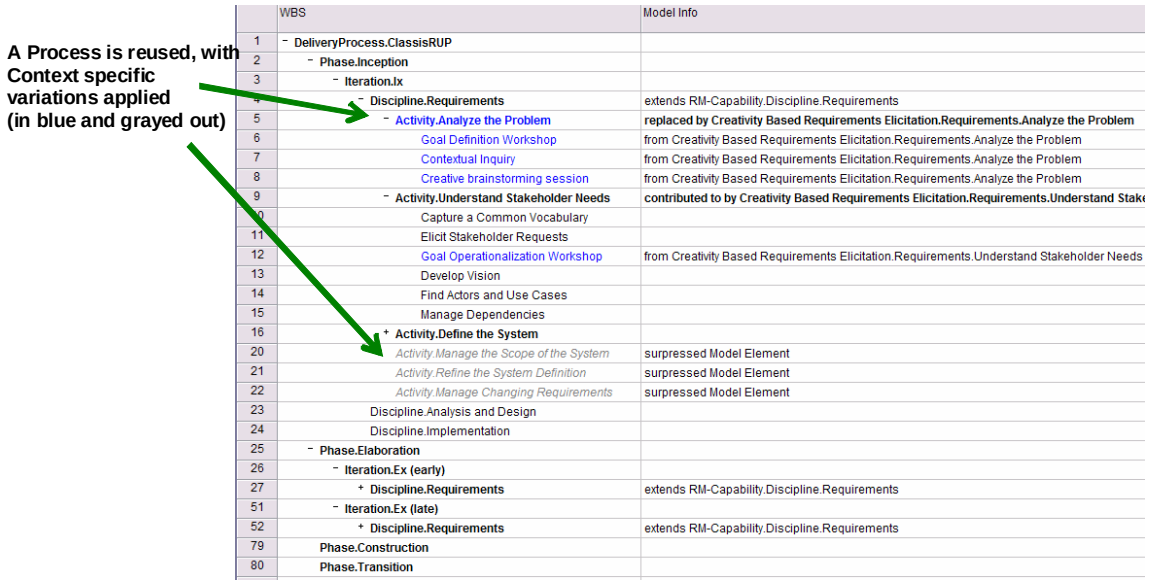


Figure 6.5 - Key terminology defined in this specification mapped to Method Content versus Process

Figure 6.5 provides an overview of how the key concepts defined in this specification are positioned to represent method content or process. Method content is primarily expressed using work product definitions, role definitions, task definitions, and guidance. Guidance, such as guidelines, whitepapers, checklists, examples, or roadmaps, are defined in the intersection of Method Content and Process, because Guidance can be defined to provide background for method content as well as for specific processes (e.g., exemplary process walkthroughs). On the right-hand side of the diagram, you see the elements used to represent processes in SPEM 2.0. The main element is the activity that can be nested to define breakdown structures as well as related to each other to define a flow of work. Activities also manage references to method content. These references are represented by matching ‘use’ concepts. Activities are used to define processes.

6.3.2 Consistent maintenance of many alternative development processes



	WBS	Model Info
1	− DeliveryProcess.ClassisRUP	
2	− Phase.Inception	
3	− Iteration.Ix	
4	− Discipline.Requirements	extends RM-Capability.Discipline.Requirements
5	− Activity.Analyze the Problem	replaced by Creativity Based Requirements Elicitation.Requirements.Analyze the Problem
6	Goal Definition Workshop	from Creativity Based Requirements Elicitation.Requirements.Analyze the Problem
7	Contextual Inquiry	from Creativity Based Requirements Elicitation.Requirements.Analyze the Problem
8	Creative brainstorming session	from Creativity Based Requirements Elicitation.Requirements.Analyze the Problem
9	− Activity.Understand Stakeholder Needs	contributed to by Creativity Based Requirements Elicitation.Requirements.Understand Stake
10	Capture a Common Vocabulary	
11	Elicit Stakeholder Requests	
12	Goal Operationalization Workshop	from Creativity Based Requirements Elicitation.Requirements.Understand Stakeholder Needs
13	Develop Vision	
14	Find Actors and Use Cases	
15	Manage Dependencies	
16	+ Activity.Define the System	
20	*Activity.Manage the Scope of the System*	surpressed Model Element
21	*Activity.Refine the System Definition*	surpressed Model Element
22	*Activity.Manage Changing Requirements*	surpressed Model Element
23	Discipline.Analysis and Design	
24	Discipline.Implementation	
25	− Phase.Elaboration	
26	− Iteration.Ex (early)	
27	+ Discipline.Requirements	extends RM-Capability.Discipline.Requirements
51	− Iteration.Ex (late)	
52	+ Discipline.Requirements	extends RM-Capability.Discipline.Requirements
79	Phase.Construction	
80	Phase.Transition	

Figure 6.6 - A process with variations: A replacement of an Activity depicted in blue color and suppressed Activities in gray color

SPEM 2.0's goal is to not only support the representation of one specific development process or the maintenance of several unrelated processes, but to provide process engineers with mechanisms to consistently and effectively manage whole families of related processes.

SPEM 2.0 utilizes an extended set of relationships for reuse and variability to realize inheritance-like and aspect-orientation-like semantics as well as concepts for process patterns and so-called method plug-ins. This allows a process engineer to maintain consistent families of processes, which on the one hand are specific to a project type and on the other hand are also variations of the same base method and process content. The results are different variants of specific processes based on the same core method content and process structures but applied with different levels of detail and scale; for example, process variants for small versus large scale development projects.

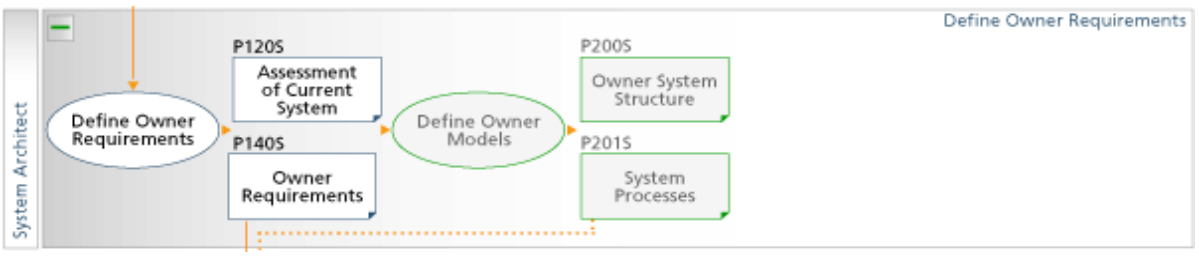


Figure 6.7 - A process with an optional step (Define Owner Models)

Figure 6.7 shows another process example (taken from Fujitsu's Macroscope) representing different variants on the same process. It depicts a process segment containing an optional process element Define Owner Models. Optional elements can be shown or hidden with the +/- toggle button in the upper left, which adds or removes it from a concrete process instance. The exact circumstances of applying or not applying the optional element are documented in the process description.

6.3.3 Many different lifecycle models

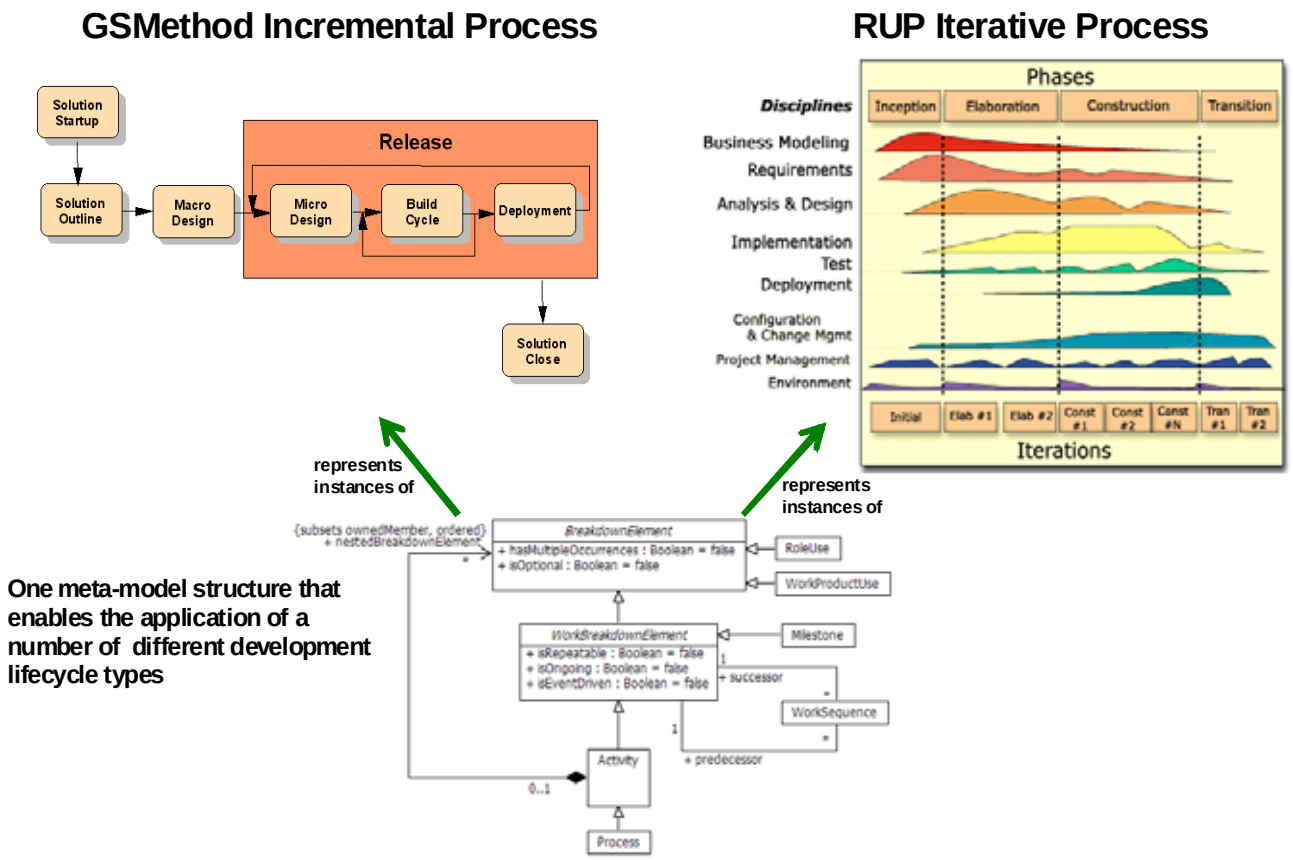


Figure 6.8 - Two processes with different lifecycle models. One common SPEM 2.0 structure to represent any of these lifecycles

A generic meta-model specification for development processes needs to be able to support different varieties and even combinations of lifecycle models such as Waterfall, Iterative, Incremental, Evolutionary, and so on for process-based planning.

The SPEM 2.0 meta-model is designed to accommodate multiple approaches. It provides a rich set of customization attributes for specifying temporal guidance for the process elements, allowing these to be mapped to project plans that are realized based on the underlying lifecycle model of the process. For example, the mapping would allow an iterative process to generate a plan that provides the user-defined number of iterations required for a specific project situation.

Additionally, SPEM 2.0 provides alternative, but consistently maintained, process presentation formalizations such as break-down structures versus workflow diagrams. This allows the Process Engineer to choose the presentation of his preference that best fits the lifecycle model used.

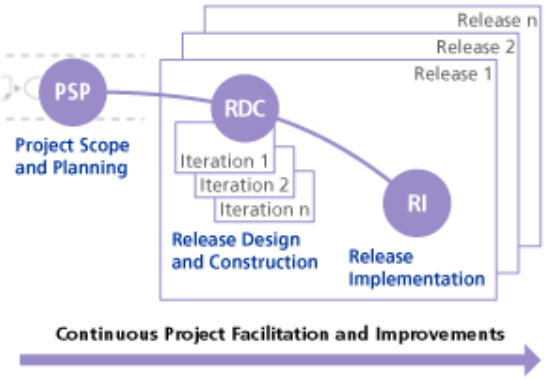


Figure 6.9 - An agile process in Macroscope. It comprises three sub-processes, each of which each follows a different lifecycle model

6.3.4 Flexible process variability and extensibility plug-in mechanism

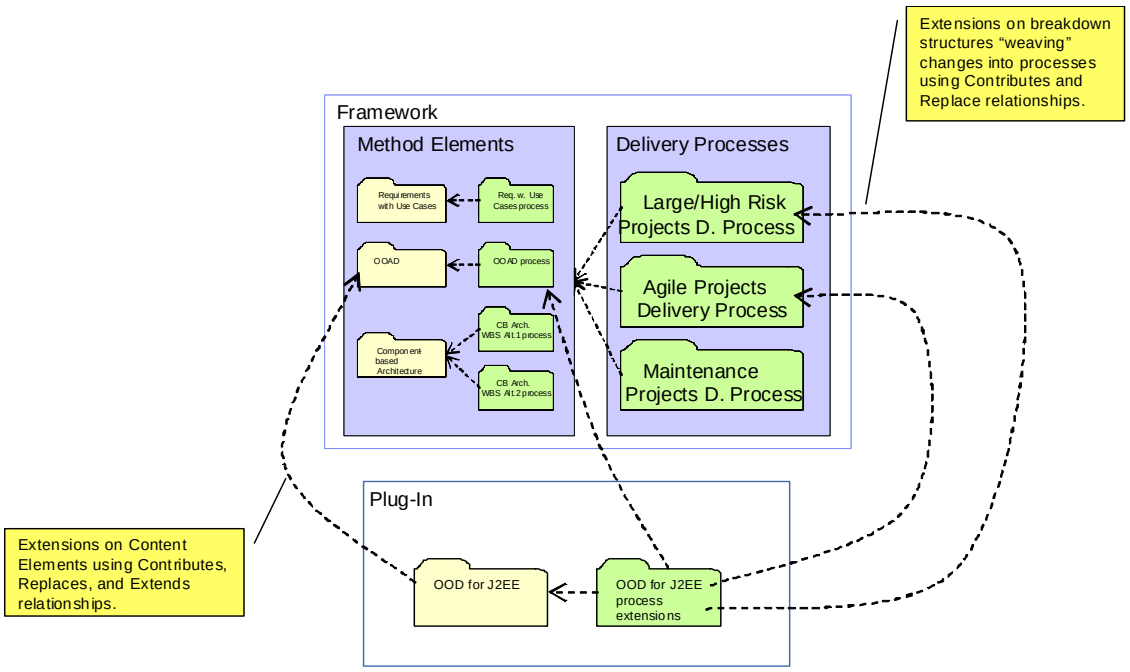


Figure 6.10 - Example for a method plug-in extending method content and processes with additional capabilities

SPEM 2.0 defines Method Plug-Ins providing capabilities for tailoring and customizing method content without directly modifying the original content. Instead, plug-ins just define the differences (contributions, replacements, suppressions) relative to the original.

SPEM 2.0 supports plug-in mechanisms for method content as well as processes represented as breakdown structures. SPEM 2.0 supports the definition of contributions to and replacements in a breakdown structure without directly modifying it, but by building a plug-in to it.

6.3.5 Reusable process patterns of best practices for rapid process assembly

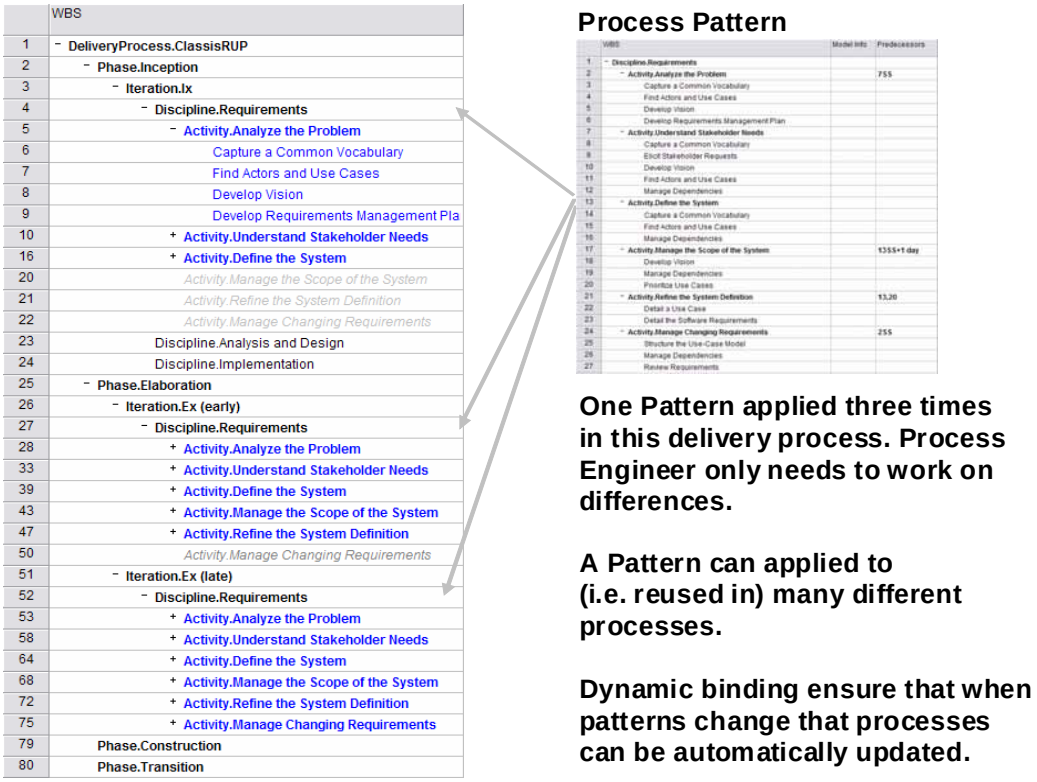


Figure 6.11 - A Process Pattern applied three times to a process; each with individual modifications

SPEM 2.0's Process Patterns are reusable building blocks for creating new development processes. Selecting and applying a Process Pattern can be done in one of two flexible ways:

- A pattern can be applied in a sophisticated copy and modify operation, which allows the process engineer to individually customize the pattern's content to his needs during the pattern application.
- SPEM 2.0 supports the application of a pattern through the Activity Use mechanisms, which is a way of reusing process structures of commonly re-occurring Activities. Activities can be factored out into patterns that can then be applied over and over again in a process. Activity Use defines relationship kinds so that when the pattern is being revised or updated, all changes will automatically be reflected in all processes that applied that pattern.

6.3.6 Replaceable and reusable Process Components realizing the principles of encapsulation

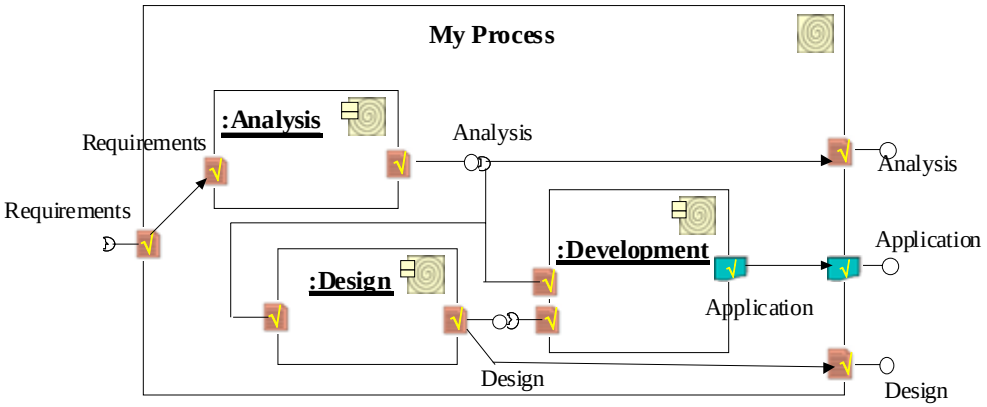


Figure 6.12 - Process Components connected via Work Product Ports

Certain situations in a software development project might require that parts of the process remain undecided or will be decided by the executing team itself (e.g., in outsourcing situations).

SPEM 2.0 provides a component concept that features ports for declaring work product input and output, allowing the user to treat the actual definition of the work that produces the outputs as a “black box.” At any point during a project, the component “realization” detailing the work can be added to the process.

The component approach also allows different styles or techniques of doing work to be replaced with others. For example, a software code output of a component could be produced with model-driven development or a code-centric technique. The component concept encapsulates the actual work and lets the development team choose the appropriate technique. It allows the team to fill the component’s realization with their choice of Activities that produce the required outputs.

6.4 Specification Formalism

This specification documents the Software & Systems Process Engineering Meta-Model 2.0 Meta-Model (SPEM 2.0 Meta-Model) and the Software & Systems Process Engineering Meta-Model 2.0 UML 2 Profile (SPEM 2.0 Profile).

The SPEM 2.0 process engineering meta-model describes the structures needed to formally express and maintain development method content and processes, i.e., it describes a language and representation schema for method contents and processes. A meta-model itself is expressed using a meta-modeling language (i.e., a meta-meta-model). The MOF standard [MOF 2.0] provides such a language by applying and extending the UML [UML 2], i.e., it describes how one would use UML 2 to describe Meta-Models. In that sense, MOF based on UML 2 is used to describe the UML 2 in a bootstrapping way.

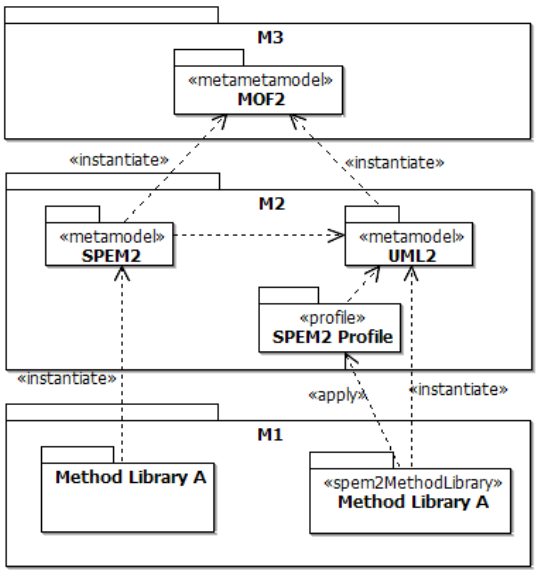


Figure 6.13 - Model Layers for UML and SPEM 2.0

The SPEM 2.0 Meta-Model is a MOF 2.0 compliant Meta-Model

This document provides a MOF 2.0 compliant meta-model specification of the SPEM 2.0 Meta-Model as depicted on the left hand side of Figure 6.13. Here you can see the different instantiation layers of the formalism used for this specification. A model defined on a higher layer defines the language to be used on the next lower layer. MOF is the universal language that can be used on any layer, but in our case MOF is instantiated from the M3 layer by SPEM 2.0 on the M2 layer. The UML 2 meta-model itself, as depicted on the right-hand side of the M2 layer, instantiates MOF2 defined on M3 layer in the same way. "Method Library A" is an example of a concrete instance of the SPEM 2.0 meta-model using SPEM 2.0 as a schema to represent its content. In that sense, "Method Library " represents a method model. For example, SPEM 2.0 defines the concepts of Roles, Tasks, and Artifacts as well as relationships between them. Method Library A on the M1 layer provides concrete instances of Roles and Artifacts from the M2 layer such as "System Analyst" and "Use Case." As you see in Figure 6.14, "Use Case" is a direct instance of the meta-class "Artifact," which is an instance of the Meta-Meta Class "Class" from the M3 layer. A use-case instance that one would create during a development project such as "Browse Catalog" for a Web-based sales system would now be an instance of the class "Use Case" on the M0 layer.

The SPEM 2.0 Meta-Model reuses parts of UML 2

The SPEM 2.0 meta-model describes all structures and attributes needed to represent SPEM 2.0-based methods and processes. However, SPEM 2.0 does not define all of its elements from scratch, but actually reuses elements from the UML 2 meta-model. Figure 6.13 shows a dependency on the M2 layer from SPEM 2.0 to UML2. This dependency expresses that parts of the SPEM 2.0 are based on definitions in the UML 2. For example, core elements of SPEM 2.0 such as "Process Element" and "Method Package" have been derived through specialization from classes from the UML 2 Infrastructure Library inheriting relationships that allow the definition of packages and packageable elements. SPEM 2.0 also reused packages from the UML 2 Superstructure, such as packages from UML 2 Activities.

The SPEM 2.0 Meta-Model has a reference implementation

The SPEM 2.0 meta-model can be directly instantiated for an implementation, i.e., a CASE tool could represent all classes from the M2 layer as Java classes and database tables. Although the SPEM 2.0 MOF meta-model is defined in UML, an instance of the model (i.e., a concrete method or process) can be represented independent of the UML.

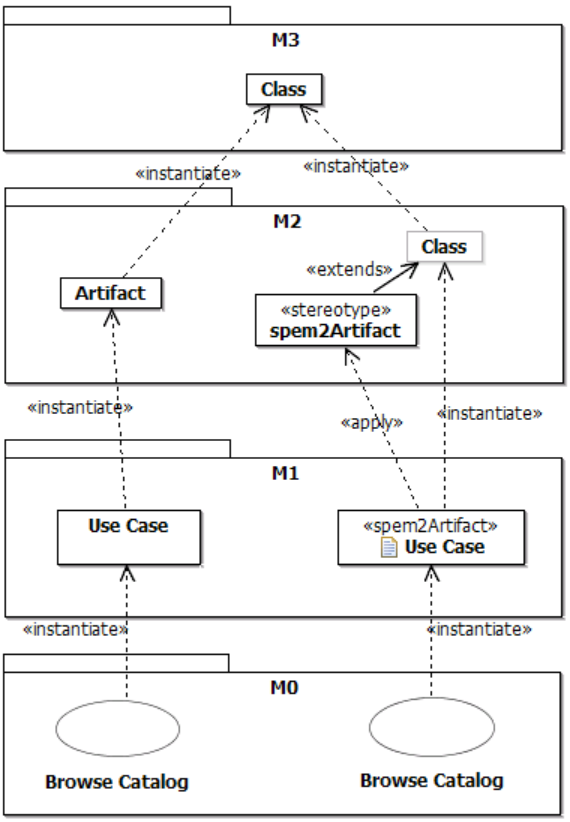


Figure 6.14 - Exemplary instantiations of the modeling layers

The SPEM 2.0 Profile is a UML 2 Profile that provides an alternative representation to the SPEM 2.0 Meta-Model

In addition to representing a method library such as “Method Library A” with these structures, creating your own implementation of these classes (e.g., using the Java classes mentioned above), one could also decide to represent classes from the M1 layer with a generic UML 2 modeling tool such as Magic Draw or IBM Rational Software Modeler. In this case, one would use UML Superstructure classes on the M2 layer and extend these with the official UML2 extension mechanisms by providing profiles with stereotypes and OCL constraints as depicted on the right hand side of Figure 6.13. For example, on the right hand side of Figure 6.14 you see a stereotype declaration for Artifact extending the UML 2 class concept. An instance on the M1 layer would be a UML 2 class that has this stereotype assigned. An M0 instance would still look the same. The only difference is the formalized representation used for the meta-model (M2) and model (M1).

The advantage of the profile approach is that one does not need to develop CASE tools to maintain methods and processes, but could use a generic UML 2 modeling tool to do so. The disadvantage of this approach is that such a UML 2 modeling tool would be very generic and that all the specific structuring rules defined in SPEM 2.0 Meta-Model as

simple relationships would need to be enforced with complicated OCL constraints, constraining the user much more than when modeling with UML. For example, restricting the number of associations a Task can have with Roles when a specific stereotype is applied enforces the simple constraint of one primary performer Role for a Task. Enforcing all basic SPEM 2.0 rules in OCL constraints would result in the tool user dealing with very generic OCL error messages and interactions, resulting in a bad user experience.

SPEM 2.0 defines a Meta-Model as well as a UML Profile

However, this specification provides definitions for both representations:

- The SPEM 2.0 Meta-Model: Defines all structures and structuring rules and is in itself complete. It has been specified as a MOF model and reuses some key classes from the UML 2 Infrastructure. It also defines the notation of specific process diagrams.
- The SPEM 2.0 UML Profile: Defines a set of UML 2 stereotypes that allows presenting SPEM 2.0 methods and processes using the UML 2. However, the definition of these stereotypes in this specification only covers their presentation, but relies on the SPEM 2.0 Meta-Model for all semantic definitions and constraints. In other words, this version of the profile does not contain any OCL constraints, but relies on the SPEM 2.0 Meta-Model semantics to define all of its constraints.

Every section defining concepts will describe both representations.

6.5 Statement of Proof of Concept and Commercial Availability

Major parts of this specification that have been implemented are freely available in open source at the Eclipse Process Framework (www.eclipse.org/epf/). See Annex C for several case studies modeling processes from different sources and domains that have been created using this technology. There are several organizations and companies that have announced they will model their free or commercial processes with this technology. Some of them are Armstrong Process Group, Open Group, Number Six Software, and Telelogic.

There is also a commercial product available from IBM that has been built on top of this open source implementation. A large number of commercial processes from IBM Software Group, IBM Rational, IBM Tivoli, and IBM Global Services as well as Sierra Systems and the DSDM Consortium have been modeled with this implementation, and are or will be available for purchase. See Annex C for examples from these processes.

Other authors of this specification such as Adaptive and Fujitsu have expressed their intent of implementing this specification and representing their processes with the concepts of this specification within a year after adoption.

6.6 Changes to Adopted OMG Specifications

This specification completely replaces the adopted SPEM specification Version 1.1, formal/05-01-06.

6.7 How to Read this Specification

Although the chapters are organized in a logical manner and can be read sequentially, this is a reference specification intended to be read in a non-sequential manner. Consequently, extensive cross-references are provided to facilitate browsing and search. Chapter 2.1 provides a detailed overview to the organization of the SPEM meta-model and its

presentation in the subsequent chapters. Chapters 7 to 14 follow that structure and present the content of the meta-model packages one by one. Chapters 15 to 18 provide additional background information for the meta-model such as examples for Process Diagrams, how to use SPEM 2.0 as a UML 2 model profile, etc.

6.8 Acknowledgements

The following companies submitted parts of this specification:

- Adaptive Ltd.
- Fujitsu & Fujtsu Consulting
- Fundacion European Software Institute
- International Business Machines Corporations
- Softeam

The following companies support this specification:

- Alcatel
- Armstrong Process Group, Inc.
- Aubry Conseil
- BAE Systems
- Borland Software Corporation
- Capgemini
- EDS
- HP
- Kabira
- Laboratoire d'Informatique Paris 6 LIP6
- Lockheed Martin
- MEGA
- MetaMatrix
- Mitre
- Number Six Software
- Sierra Systems
- SINTEF ICT
- Telelogic
- Unisys

The specification was written by Peter Haumer (IBM Rational Software, phaumer@us.ibm.com) with direct contributions by Todd Fredrickson (IBM Global Services, toddfr@us.ibm.com) and Philippe Desfray [Process Components] (Softeam, Philippe.Desfray@softeam.fr). It incorporates additional concepts and direct feedback from Kamal Ahluwalia (Osellus), Chris Armstrong (Armstrong Process Group), Claude Aubry (Aubry Conseil), Reda Bendraou (Laboratoire d'Informatique Paris 6 LIP6), Philippe Desfray (Softeam), Tony Di Salvo, Serge Deschamps (Fujitsu Consulting), Karl Frank (Borland), Kenneth Hussey (IBM Rational), Sridhar Iyengar (IBM Rational), Xabier Larrucea (ESI), Hiroshi Miyazaki (Fujitsu), Pete Rivett (Adaptive).

The authors would also like to thank Eric Aker, Ricardo Balduino, Alfredo Bencomo, Stefan Bergstrom, Amanda Brijpaul, James Bruck, Trevor Collins, Michael Darmody, Phil Edwards, Carl Engel, Jonathan Foo, David Gerber, Daniel Giraudeau, Carlos Goti, Bjorn Gustafsson, Michael Hanford, Margaret Hedstrom, John Hollingsworth, Sigurd Hopen, Kelli Houston, Volker Kopetzky, Mark Krol, Per Kroll, Phong Le, Guillermo Lois, Kelvin Low, Brian Lyons, Bruce MacIsaac, Jas Madhur, Sumeet Malhotra, Jason Xabier Mansell, Andrew Mirsky, Erik O'Neil, Russell Pannone, Thierry Paradan, Steve Pendergrass, Cecile Peraire, Dan Popescu, Brian Schlosser, Gordon Schneemann, John Smith, Ian Spence, Vivienne Suen, Shilpa Toraskar, Bobbi Underbakke, Gerard Vahee, Steve Von Wald, Jinhua Xi, Bingxue Xu, Charles Yan as well as many other process stakeholders and reviewers.

7 Using SPEM 2.0 as a UML 2 Superstructure Profile

As outlined in Section 6.4, “Specification Formalism,” on page 19 this specification defines the Software Process Engineering Meta-Model 2.0 as a MOF 2.0-based Meta-Model (referred to as the ‘SPEM 2.0 Meta-Model’) as well as a UML 2 Superstructure-based Profile (referred to as the ‘SPEM 2.0 Profile’). Therefore, every concept definition in this specification defines in detail the meta-model classes as well as the UML Stereotypes. However, because the focus of the descriptions throughout Chapters 8 to 14 is on the meta-model, this chapter provides a general overview, rationale, and summary for the SPEM 2.0 Profile. In other words, it condenses the definitions distributed over the different chapters into one coherent overview. The chapter assumes general knowledge of the key meta-model concepts defined in Chapters 8 to 14. “Annex A: SPEM 2.0 UML 2 Profile Summary” provides the complete alphabetical overview to all of these stereotypes.

The specification defines in addition to the SPEM 2.0 Profile a second non-standard convenience profile called “SPEM 2.0 Base Plugin Profile,” which provides useful stereotypes for many typical Kind (Section 8.2) class instances. Many examples presented in this specification use this profile as well. See Chapter 18 for the definition of this profile.

7.1 SPEM 2.0 Profile Overview

The SPEM 2.0 Profile defines stereotypes for UML 2 Superstructure model elements for almost every meta-model concept introduced in the following chapters that can be represented in UML 2 diagrams in a meaningful way. Every stereotype definition is listed in each section defining the concept and summarized again in an alphabetical list in Annex A. All of these stereotypes can be applied as extensions to the specified UML 2 meta-model classes as specified for each SPEM 2.0 meta-model class in the subsection “UML 2 Profile Notation.” Every stereotype is defined throughout Chapters 8 to 14 in a UML 2 stereotype diagram as well as in a table as the one shown below that lists

- the stereotype name,
- the name of the metaclass it extends as well as the names of superclass stereotypes it is derived from,
- the keyword that is used for presentation of the stereotype,
- the names of properties that it defines,
- a statement indicating if the stereotype is abstract, and
- a graphical presentation icon.

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
TaskDefinition	MethodContentElement, WorkDefinition / Classifier	task definition	n/a	no	

Figure 7.1 shows an example with a UML Class Diagram that represents several SPEM 2.0 concepts as UML classes with their respective stereotypes applied.

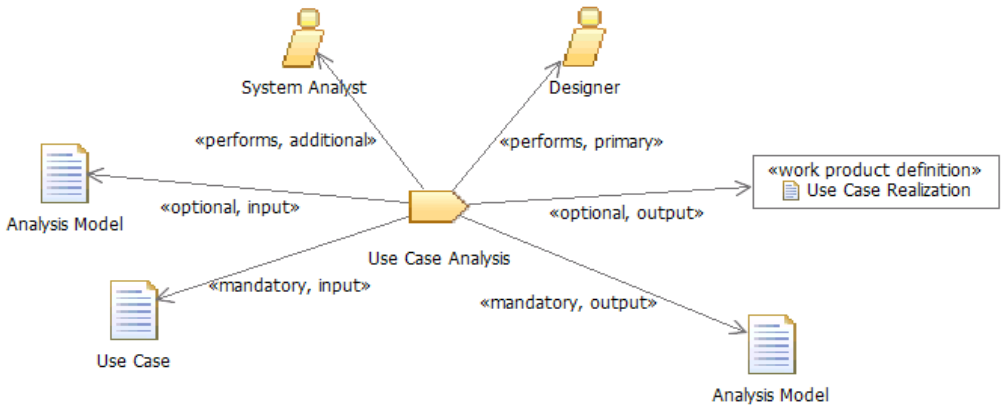


Figure 7.1 - UML 2 Class Diagram using SPEM 2.0 Profile with stereotyped classes

The example shows that the Task Definition “Use Case Analysis” has mandatory and optional input and output types, as well as being performed by two Role Definitions: Designer, being the primary performer as well as System Analyst, being the additional or optional performer. Note that the stereotypes ‘primary’ and ‘additional’ are actually not part of the SPEM 2.0 UML Profile, but have been defined as Kinds of the Task Definition Performer and are therefore part of the SPEM Base Plugin defined in Chapter 18.

Figure 7.1 also provides an example for how stereotypes can be presented in different ways. The default presentation option used for almost all diagrams created for this specification is to define and use an icon for every stereotype defined for classes derived from the UML 2 Classifier. In addition, stereotypes have textual presentations. Figure 7.1 shows examples for using the icon presentation as well as one example (Use Case Realization) that is presented using the textual stereotype presentation and an iconic decoration.

For backwards compatibility to SPEM 1.1 models it is also valid to use stereotype icons defined in the 1.1 version of this specification. Figure 7.2 shows the same classes of Figure 7.1 but with the SPEM 1.1 icons. All of these alternative backwards-compatible icons are listed with the classes they apply to. Note that SPEM 2.0 introduces several new concepts and that not all SPEM 2.0 concepts have mapping version 1.1 stereotypes.

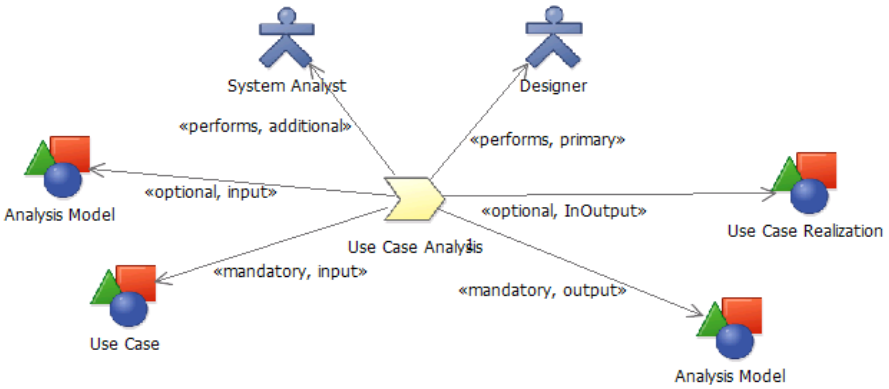


Figure 7.2 - The same SPEM 2.0 class diagram of Figure 7-1 using SPEM 1.1 stereotype icons

It is also important to point out that the diagram of Figure 7.2 does actually not represent valid SPEM 1.1 syntax. In SPEM 2.0 Work Definitions such as Tasks and Activities are represented as stereotyped UML 2 Activities. SPEM 1.1 required Work Definitions to be modeled as operations of Roles and Process Performers. Therefore, the correct SPEM 1.1 equivalent of Figure 7.2 is depicted in Figure 7.3 for reference. It shows the Designer owning the Use Case Analysis Work Definition as an operation.

Figure 7.3 - Syntactically correct SPEM 1.1 representation of Figure 7-2

The examples of Figure 7.2 and Figure 7.3 also show the limitations of SPEM 1.1, in which it was not possible to show assisting (or additional performing) roles as part of a class diagram. They had to be specified via tagged values and/or annotated with notes. Assisting roles could have been specified in Use Case diagrams in SPEM 1.1, but this approach caused a lot of confusion because the SPEM semantics heavily contradicted the UML use case semantics. Use Case diagrams have therefore been invalidated for the SPEM 2.0 Profile. Specific reasons were that SPEM 1.1 applied the Activity stereotype to UML use cases, which was confusing as use cases were derived from classifier in UML and not operation as SPEM 1.1 activities. Also the use of UML actors for SPEM 1.1 Roles contradicts UML semantics as SPEM Roles are not external interacting entities as required for UML actors, but are the actual performers and therefore part of the behavior described by SPEM activities. Finally, it was also not possible to specify in SPEM 1.1 if a parameter was optional or mandatory as can be done with new SPEM 2.0 stereotypes for modeling inputs and outputs for a Task or Activity.

The SPEM 2.0 Profile defined in this specification focuses on utilizing Class, Activity, and State Model Diagrams. All proprietary diagrams defined in this specification such as the Work Product Dependency diagram (Section 13.15, class diagram) or the Team Profile diagram (Section 13.15, class diagram) can be represented with any of these three diagram types. The use of other UML 2 diagram types is not discouraged, but has not been considered in this release.

The use of these stereotypes for the UML diagrams has to be restricted to the rules defined by the SPEM 2.0 meta-model. It is, for example, not valid to define a Step that is composed of Task. Although, UML 2 would allow you to model such a construct, it would not represent a valid SPEM 2.0 model. However, this version of the specification does not define detailed OCL constraints formulating the exact rules for using the SPEM 2.0 Profile stereotypes. The rules would be redundant with the information already captured in the SPEM 2.0 meta-model and would not provide additional value (except for supporting UML 2 CASE tool vendors in implementing the SPEM 2.0 Profile).

The following sections define more specific principles and applications of the SPEM 2.0 Profile stereotypes defined throughout this specification.

7.2 Describing Work Definitions and Work Breakdown as UML Behavior Models

The Software Process Engineering Meta-Model 2.0 (SPEM 2.0) separates reusable Method Content from its application in Processes (see Section 6.3.1, “Clear separation of method content definitions from the development process application of method content,” on page 12 and Chapter 13). For example, a Task Definition (Section 12.9) defined as Method Content can be invoked many times throughout a development process. Each invocation is defined with an individual Task Use (Section 13.14), which can manage its own invocation specifics such as a change in the Roles involved in performing the Task or an omission of specific work product input types.

The UML 2 Superstructure specification defines a very similar approach for representing behavior. It allows defining reusable behavior with UML Activities that are invoked by Actions presented in Activity diagrams. The behavior expressed by UML Activities can either be bound to objects and related to the invocation of specific object’s operations by Actions, but also be defined independent of objects as so-called ‘first class behavior.’ Further, UML Activities can be nested. In other words, a UML Activity can be defined by workflow of Actions, which in turn can invoke other Activities, etc. Finally, UML Activities derive from the UML Classifier and can therefore have structural relationships such as association to other UML Classes. All of the UML 2 capabilities described in this paragraph have been utilized when representing sub-classes SPEM 2.0 Work Definitions (Section 8.4) in the SPEM 2.0 Profile.

Work Definitions (Task, Step, Activity) are represented by UML 2 Activities

The Work Definition’s Task Definition and Step (Section 12.8) specify ‘first class behavior’ describing development work. These descriptions define the behavior of either one Role Definition performing the work or many Role Definitions participating and collaborating with each other during the Task Definition’s performance. On the other hand, formalizing the exact behavior or collaboration of Role Definitions, in addition to defining Steps for a Task Definition and describing them in natural language, is often regarded as over-specification and not common practice. Therefore, additional UML Collaborations expressing Role Definition interactions are not encouraged by this specification. Instead it is recommended to describe Task Definition and Steps, the involvement of specific Roles, as well as their collaboration needs, with textual descriptions using the mainDescription attribute defined in Section 11.2.

As a result, Task Definitions and Steps extend UML 2 Classifier and can be represented as UML 2 Activities with respective stereotypes (defined in Section 12.8 and Section 12.9). A Task Definition is represented as an Activity which contains a diagram with CallBehavior Actions that invoke Steps represented as UML Activities as well, which are owned by the Task’s UML Activity. Steps can be refined into sub-Steps, which are represented as owned UML Activities as well.

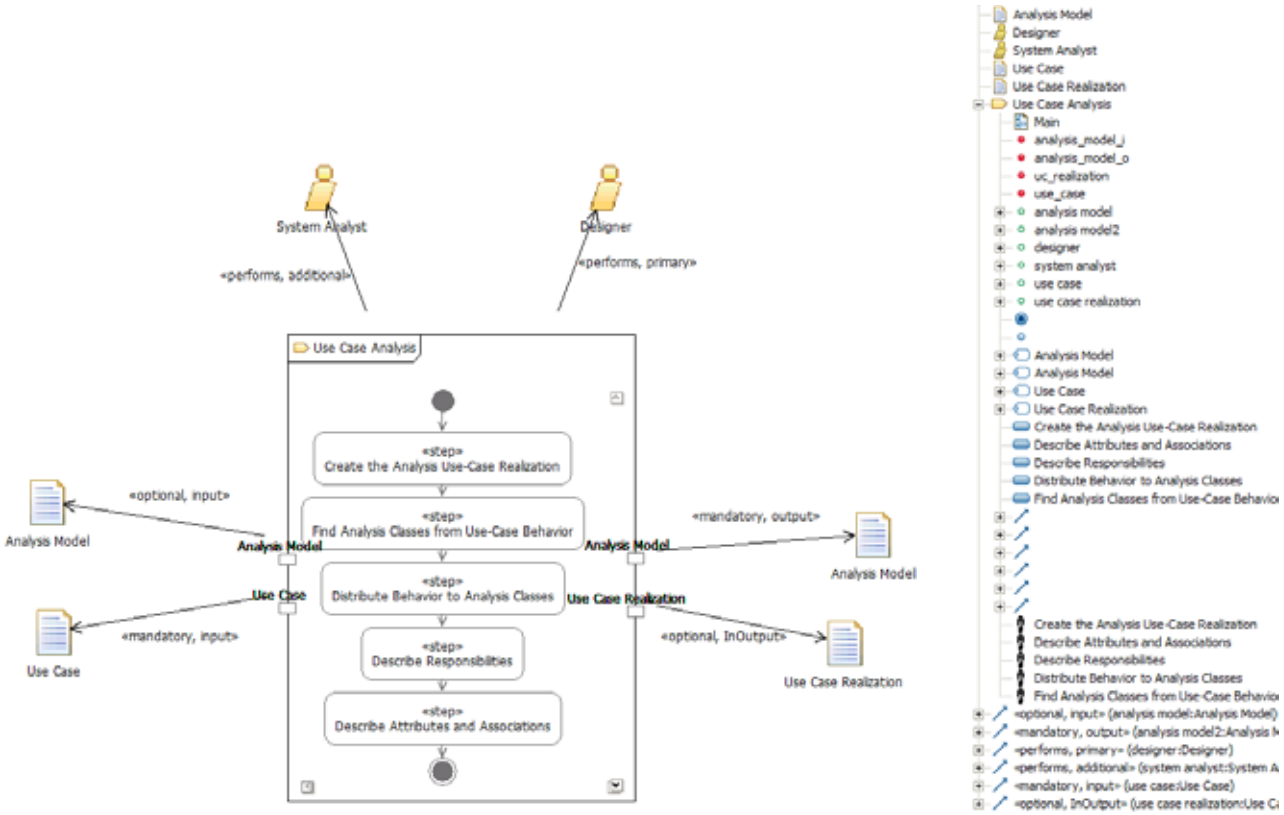


Figure 7.4 - Task represented as stereotyped UML 2 Activity diagram (left) and model objects needed (right) showing Steps as CallBehavior Actions invoking Activities (step icons with the same name) and performing Roles

Figure 7.4 shows the “Use Case Analysis” example from Figure 7.1 providing a more detailed representation for the Task, which has been now described using Steps. The picture shows on the left a UML 2 Activity diagram nested in a UML 2 class diagram with stereotyped model elements. It shows on the right an overview to the model objects created for this diagram. Figure 7.4 shows the Steps as Actions that invoke Activities with the same name. It further represents the primary performing role System Analyst as well as the additional performing role Designer. Further, it defines Activity Parameters which refer to the Artifacts Analysis Model, Use Case Model, and Use Case Realization.

Processes represent Task Uses with Actions

When defining a Process with breakdown structures, two additional SPEM 2.0 meta-model concepts are used to represent work:

- SPEM Activities (Section 13.8), which are special Work Definitions and can be represented as UML Activities just as any other SPEM 2.0 Work Definition. Activities can be nested and they can contain any Method Content Use element (Section 13.7) in particular Task Uses.
- Task Uses (Section 13.14), which are not Work Definitions by themselves, but refer to Tasks and represent invocations of Task in the Process. Therefore, Task Uses are represented in the SPEM 2.0 Profile as stereotyped Actions.

Figure 7.5 provides an example of a SPEM 2.0 Activity which has been modeled as a UML Activity. The Activity diagram shows Actions representing SPEM 2.0 Task Uses. Input and Output pins can be used to specify the specific input and output Work Product Uses needed for these specific Task Uses for this specific Activity. For example, although the Task “Use Case Analysis” has been defined in Figure 7.5 to have an Analysis Model input, this particular Task Use overrides the Task Definition in this particular Activity to not require this input by not specifying such an input pin. See Section 13.7 for more details and example for Activity-local changes defined with Method Content Uses.

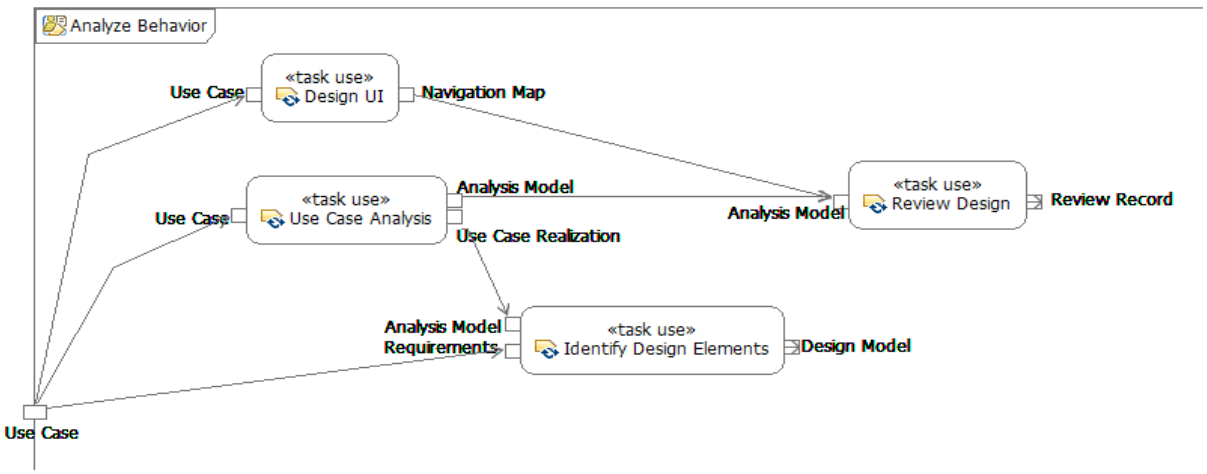


Figure 7.5 - Example for an Activity with Task Use

UML Activity Diagrams can be used to represent nesting of SPEM 2.0 Activities defining workflow and the breakdown of work.

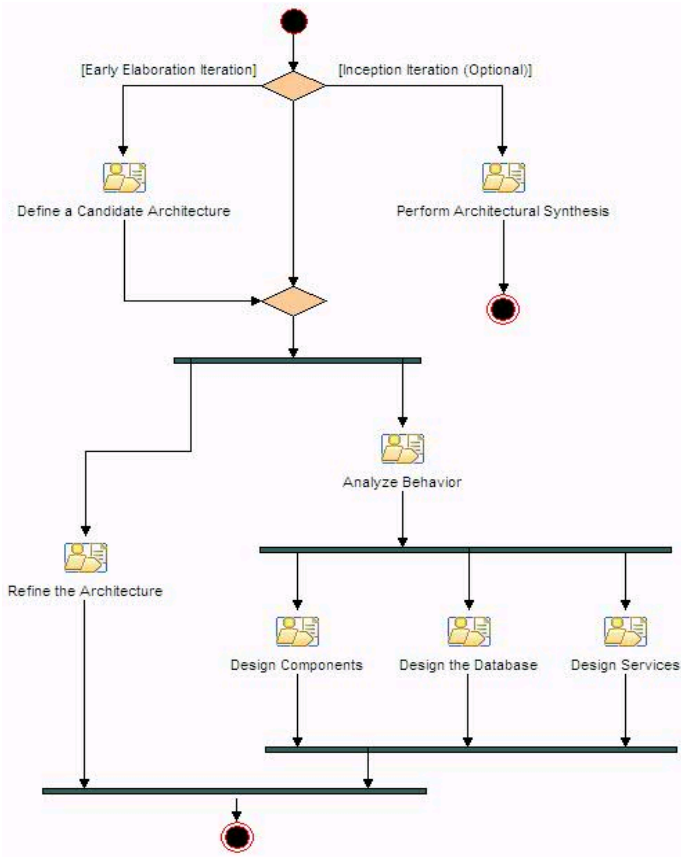


Figure 7.6 - SPEM 2.0 Activity Workflow represented as UML Activity diagram

7.3 Describing Work Product Evolution with State Machines

In addition to or instead of modeling the flow of work with UML 2 Activity diagrams, many process engineers focus on describing the ‘flow of state’ that many key development work products go through. For example, Figure 7.7 shows on the left-hand side two typical ‘lifecycles’ for two artifacts.

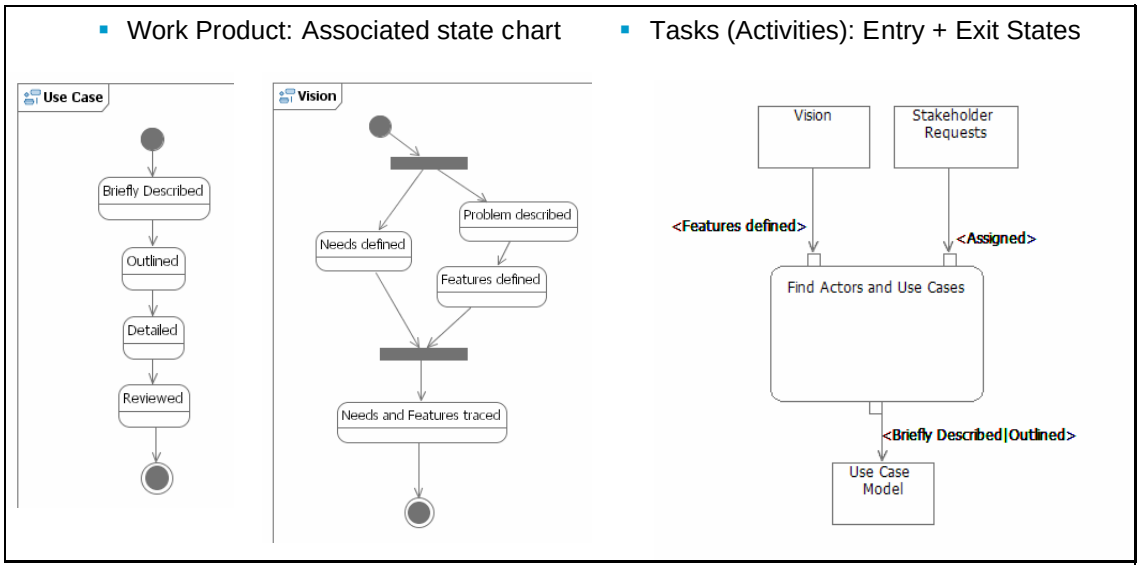


Figure 7.7 - Examples for work product state machines (left) and how states can be used in object flow diagrams to qualify the inputs and outputs of activities

A use case, for instance, is typically only briefly described when identified and modeled. During use case workshops, interviews, or other collaborations it will be outlined to gain a better understanding and agreement of the use case's main flows. Later it is detailed and reviewed. Using the SPEM 2.0 UML Profile UML 2 State Machines can be utilized to define and model such work product lifecycles as depicted in Figure 7.7 (left).

As one can already see from the description of these states for use case example in the text above, it is quite straight forward to link the state transitions to work definitions that actually cause the state transition. Hence, the right-hand side of Figure 7.7 shows how the SPEM 2.0 UML Profile extends UML 2 Activity diagrams with the ability to annotate states to input and output PINs. Thus, state machines and activity diagrams can be related to each other such that activities can cause a state transition for one or more work products. In the case of Figure 7.7, we see three work products participating in the activity "Find Actors and Use Cases." The diagram specifies that the states listed for the inputs are to be interpreted as a precondition for both the activity to be performed and the state listed for the Use Case Model output (the state the activity is supposed to establish for the work product before it can be considered to be finished). See Section 10.1 on how these so-called entry and exit states for work definitions have been modeled in the SPEM 2.0 meta-model.

7.4 Relating Work Product State to Work Definitions

As we have seen in the last section, transitions of work product state models can be systematically linked to work definitions. In that sense, processes can be described entirely from a work product lifecycle's perspective and still comprise all the information that breakdown structure oriented processes provide, such as work definitions, roles, and guidance.

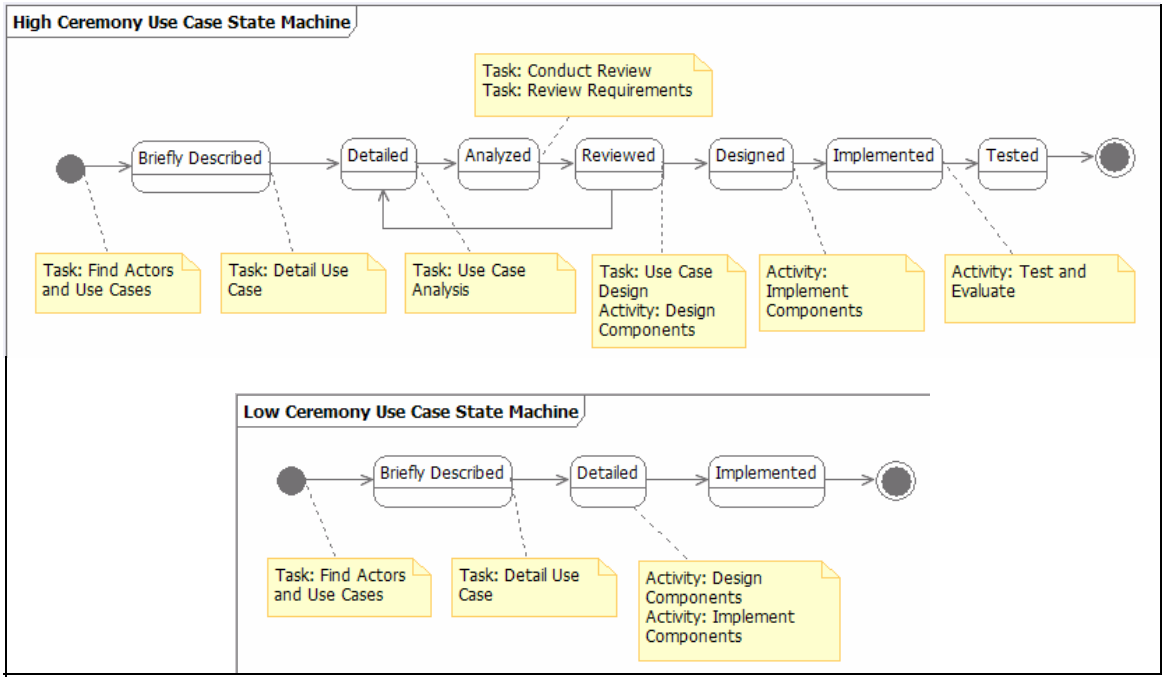


Figure 7.8 - State transition annotated with mapping work definitions

Figure 7.8 provides two more examples using UML 2 state models with the SPEM 2.0 UML profile for the Use Case artifact. These examples define two alternative ways of working with use cases that reflect alternative processes for process users to choose from when instantiating a process, such as processes that require high ceremony ways of working and artifact management as well as low ceremony ones. Each transition is annotated with reference to work definitions such as Tasks or Activities that define how the use case would be brought from one state to another state. Thus, the task Find Actors and Use Cases would result in use cases in the state of Briefly Described, whereas the task Detail Use Case assumes/requires this state on a use case and transitions it to the state Detailed. Transitions can be linked to one or more work definitions such as a Step, Task, Activity, or Process.

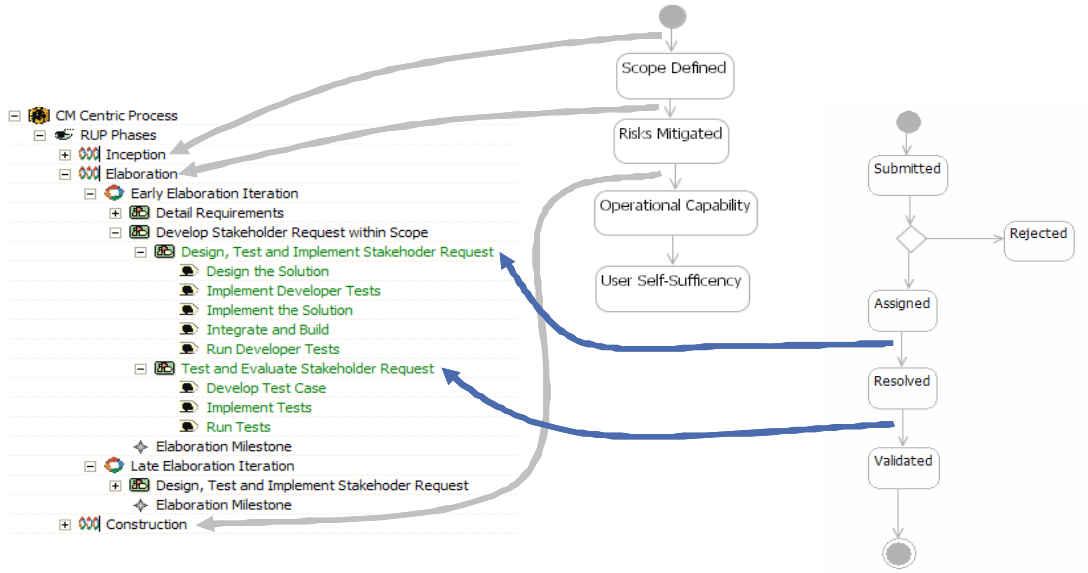


Figure 7.9 - Another view of how state transitions relate to work definitions

Systematically utilizing the relationships between states, transitions, and work definitions can help process engineers in finding the right work definitions when developing the process starting with work products and their state models. To that extent, Figure 7.9 provides another view on how state transitions and work definitions relate to each other. It shows how state transitions can be directly mapped to processes represented as breakdown structures. Different levels of the breakdown structure relate to different work products. For example, the top-level phases of the process shown in Figure 7.9 link to a work product that tracks the state of the overall project, while state transitions for a work product (such as defect) link to low-level work definitions in the breakdown structure.

8 Core

SPEM 2.0 defines several compliance points that enable implementers to freely choose among the process modeling capability they need by selecting specific meta-model combinations for implementation (see Section 2.2, and Section 2.6 for details). The Core meta-model package, however, contains the meta-model classes and abstractions that build the foundation for all other meta-model packages. In other words, all the common classes among all compliance levels defining the core of SPEM 2.0 have been placed in this meta-model package.

Core defines classes for two SPEM 2.0 capabilities:

- The ability for a SPEM 2.0 user to create user-defined qualifications for a SPEM 2.0 class allowing users to distinguish different ‘kinds’ of SPEM 2.0 class instances.
- A set of abstract classes to define work expressed as SPEM 2.0 processes. All SPEM 2.0 classes that derive from these classes are intended to map to behavior classes of behavior models (e.g., can be assigned as stereotypes to UML 2 Activities (as in Chapter 7) or linked to behaviored classifiers (as in Chapter 9)).

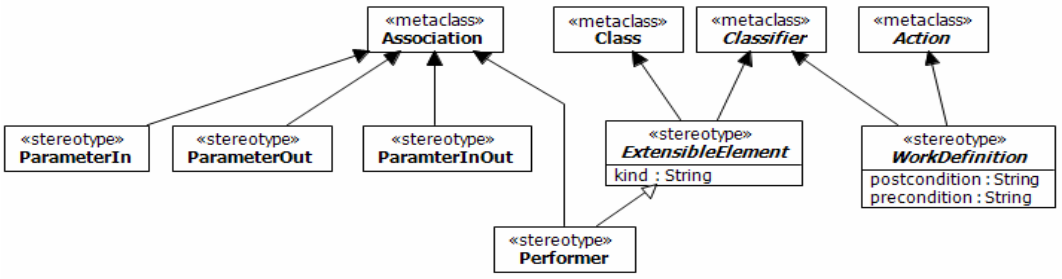


Figure 8.1 - The SPEM 2.0 UML 2 Profile stereotypes defined in the Core package

8.1 Extensible Element

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

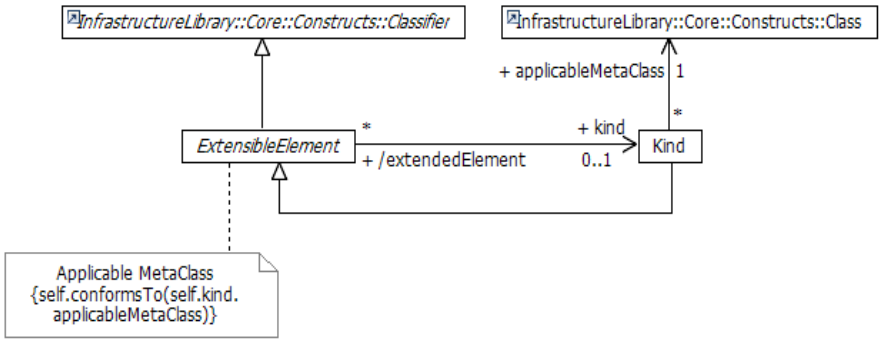


Figure 8.2 - Extensible Elements can have a Kind

Description

Extensible Element is an abstract generalization that represents any SPEM 2.0 class for which it is possible to assign a Kind to its instances expressing a user-defined qualification. Every SPEM 2.0 class that allows such a qualification derives directly or indirectly from Extensible Element.

Association Properties

- kind: Kind — An instance of Extensible Element can be linked to zero or one Kind in which the Kind instance expresses a specific user-defined qualification for that Extensible Element instance.

Semantics

Extensible Element provides the property of relating a Kind class to its sub-classes. Such Kinds cannot be reused for many different subtypes of Extensible Element and therefore can only be related to exactly one meta-model class.

Extensible Element defines a constraint called Applicable MetaClass. The constraint defines that the Kind an instance of Extensible Element links to has to link to an applicable Meta Class that is either of the same class as Extensible Element or a superclass of the Extensible Element class. See Semantics of Kind (Section 8.2) for more details.

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
ExtensibleElement	Classifier	extensible element	kind	yes	

8.2 Kind

Super Class

Extensible Element

Description

Kind is an Extensible Element. It instances are used to qualify other SPEM 2.0 Extensible Element instances with a user-defined type or kind.

Association Properties

- applicableMetaClass: Class — An instance of Kind can only be used for instances of exactly one SPEM 2.0 Extensible Element subclass or its subclasses. This property specifies which one.

Semantics

As many methods and processes to be modeled with SPEM 2.0 need to define their own refined vocabulary, Kind provides the ability to a SPEM 2.0 modeler to express such user-defined qualifications for instances of Extensible Element. Because Kind is an Extensible Element itself one can define Kinds for the Kind class itself as well.

For example, a subclass of Extensible Elements that typically utilizes Kinds is the meta-model class Guidance (Section 11.4). Typical Guidance kinds would be: White Paper, Guideline, Checklist, Template, Reports, etc. Because of the Applicable MetaClass constraints, these Kinds can only be related to instances of the Guidance class as well as instances of any subclasses of Guidance such as Metric (Section 11.5).

Other examples are Kinds for Breakdown Elements such as Phase, Iteration, Sprint, or Increment; Kinds of Work Products such as Artifact, Deliverable, or Outcome; Kinds of Categories such as Classification, View, Grouping, Practice, Discipline, Domain, etc. All of these allow SPEM 2.0 users to distinguish the special kinds of SPEM 2.0 elements that play an important role in their method or process. See Chapter 18, which defines the SPEM 2.0 base plug-in containing a collection of typical predefined Software Engineering Kinds for many more examples.

SPEM 2.0 Profile Notation

Modeled as a string property of Extensible Element (Section 8.1).

8.3 Parameter Direction Kind

Super Class

n/a : Enumeration

Description

This enumeration defines for Work Definition Parameter instances whether the parameter represents an input, output, or input as well as output.

Enumeration Literals

- in — A Work Definition Parameter instance with this direction value represents an input.
- out — A Work Definition Parameter instance with this direction value represents an output.
- inout — A Work Definition Parameter instance with this direction value represents an input and output.

SPEM 2.0 Profile Notation

See Work Definition Parameter (Section 8.5).

8.4 Work Definition

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

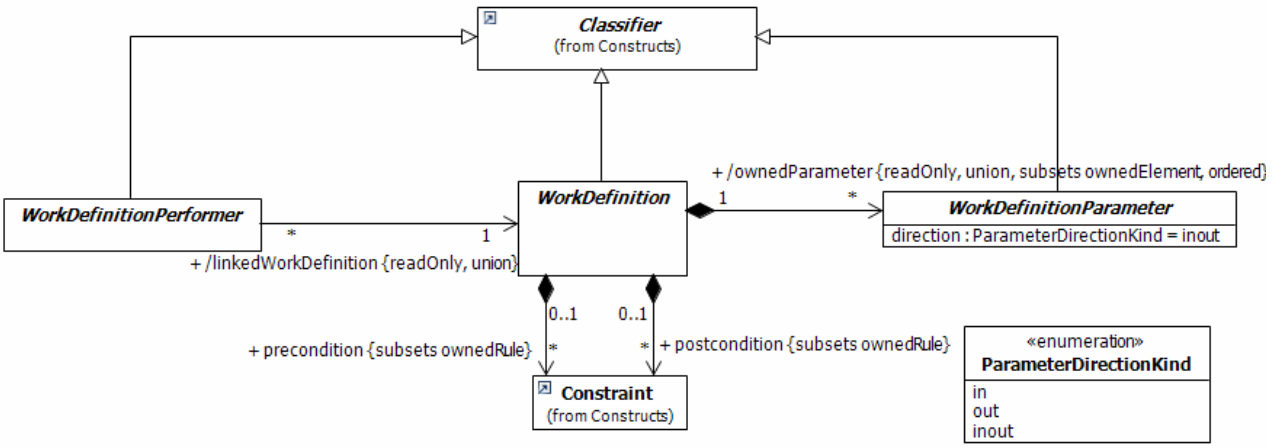


Figure 8.3 - Work Definition and its related elements

Description

Work Definition is an abstract Classifier that generalizes all definitions of work within SPEM 2.0. Work Definition defines some default associations to Work Definition Parameter and Constraint. Work Definitions can contain sets of pre- and post-conditions defining constraints that need to be valid before the described work can begin or before it can be declared as finished. Note that general UML constraints inherited via Classifier can be used to define additional constraints and rules for Work Definitions.

Association Properties

- /ownedParameter: Work DefinitionParameter

  Work Definition can define an ordered set of parameters to specify inputs and outputs. The concrete subclasses of Work Definition need to define their own subclasses of Work Definition Parameter to add reference to concrete input/output meta types.

- precondition: Constraint

  This composition association adds an optional pre-condition to a Work Definition. A pre-condition defines any kind of constraint that must evaluate to true before the work described for the Work Definition can start. For example, a pre-condition could define that an input Work Product needs to be in a specific state or that other related work must be in a certain state (e.g., 'Input document X has been reviewed and signed by customer AND the work defined by Work Definition "Management Review" is complete') before the work can begin.

- postcondition: Constraint

  This composition association adds an optional post-condition to a Work Definition. A post-condition defines any kind of constraint that must evaluate to true before the work described for the Work Definition can be declared completed or finished and which other Work Definitions might depend upon (e.g., for their pre-conditions). For example, a post-condition could define that a work product defined to be the output must be in a specific state before the Work Definition can end (e.g., 'Use Case must be in state fully described and reviewed by System Analyst').

Semantics

Work Definitions represent the behavior for doing work. In contrast to SPEM 1.1 the Work Definition behavior is not bound to one specific classifier as an operation, but is a performer independent definition of work represented as a classifier itself. For example, a Work Definition could represent work that is being performed by one specific Role (e.g., a Role performing a specific Activity), by many Roles working in close collaboration (many Roles all working together on the same interdisciplinary Activity), or complex work that is performed throughout the lifecycle (e.g., a process defining a breakdown structure for organizing larger composite units of work performed by many Roles working in collaboration).

Work Definitions can be modeled with behavior model representations such as UML 2 Activity diagrams by merging this UML 2 meta-model package into the Process Behavior package. However, it is also a common practice to just textually describe the behavior of work definitions. Such descriptions can be added by merging this package into the Managed Content package.

SPEM 2.0 Profile Notation

Stereotype	Metaclass	Keyword	Properties	Abstract	Icon
WorkDefiniton	Classifier, Action	n/a	precondition, postcondition	yes	n/a

8.5 Work Definition Parameter

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

Description

A Work Definition Parameter is an abstract generalization for Process Elements that represent parameter for Work Definitions. It is used for declarations of inputs and outputs. The meta-classes for the input/output types are to be defined by Work Definition Parameters concrete subclasses.

Attributes

- direction: ParameterDirectionKind = inout — This attribute represents the kind of the input as specified by the enumeration Parameter Direction Kind.

Semantics

This class is to be specialized by a class defining an association to the class that defines the input/output meta-class. SPEM 2.0 distinguishes between Work Product Definitions and Work Product Use. The general parameter class Work Definition Parameter defines common properties used by Work Definition specialization that either use Work Product Definitions or Work Product Uses.

SPEM 2.0 Profile Notation

The UML 2 Profile does not represent Work Definition Parameter instances as classes, but creates association between Work Definitions and Work Products using the following stereotypes to indicate the Parameter Direction Kind. For simplification, the stereotype is not abstract. All subclasses of Work Definition Parameter use the same keywords.

Stereotype	Metaclass	Keyword	Properties	Abstract	Icon
ParameterIn	Association	input	n/a	no	n/a
ParameterOut	Association	output	n/a	no	n/a
ParameterInOut	Association	inoutput	n/a	no	n/a

8.6 Work Definition Performer

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

Description

Work Definition Performer is an abstract Classifier that represents the relationship of a work performer to a Work Definition. Different specialization of Work Definition will introduce different kinds of performers. Work Definition Performer is intended to be specialized adding the association to the concrete performer meta class.

Association Properties

- /linkedWorkDefinition: WorkDefinition — This derived union provides access to all the Work Definitions a Work Definition Performer instance is related to.

Semantics

No additional semantics. See its subclasses.

SPEM 2.0 Profile Notation

The UML 2 Profile represents Work Definition Performer as concrete association extension. The different Kinds of performers shall be modeled with additional stereotypes.

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
Performer	ExtensibleElement / Association	performs	n/a	no	n/a

9 Process Structure

The Process Structure meta-model package contains the basic structural elements for defining development processes. Development processes define how development projects shall be executed. One of the most common characteristics found within the many different definitions of process in literature is sequencing of phases and milestones expressing a lifecycle of the product under development. Processes also define how to get from one milestone to the next by defining sequences of work, operations, or events that usually take up time, expertise, or other resource and which produce some outcome.

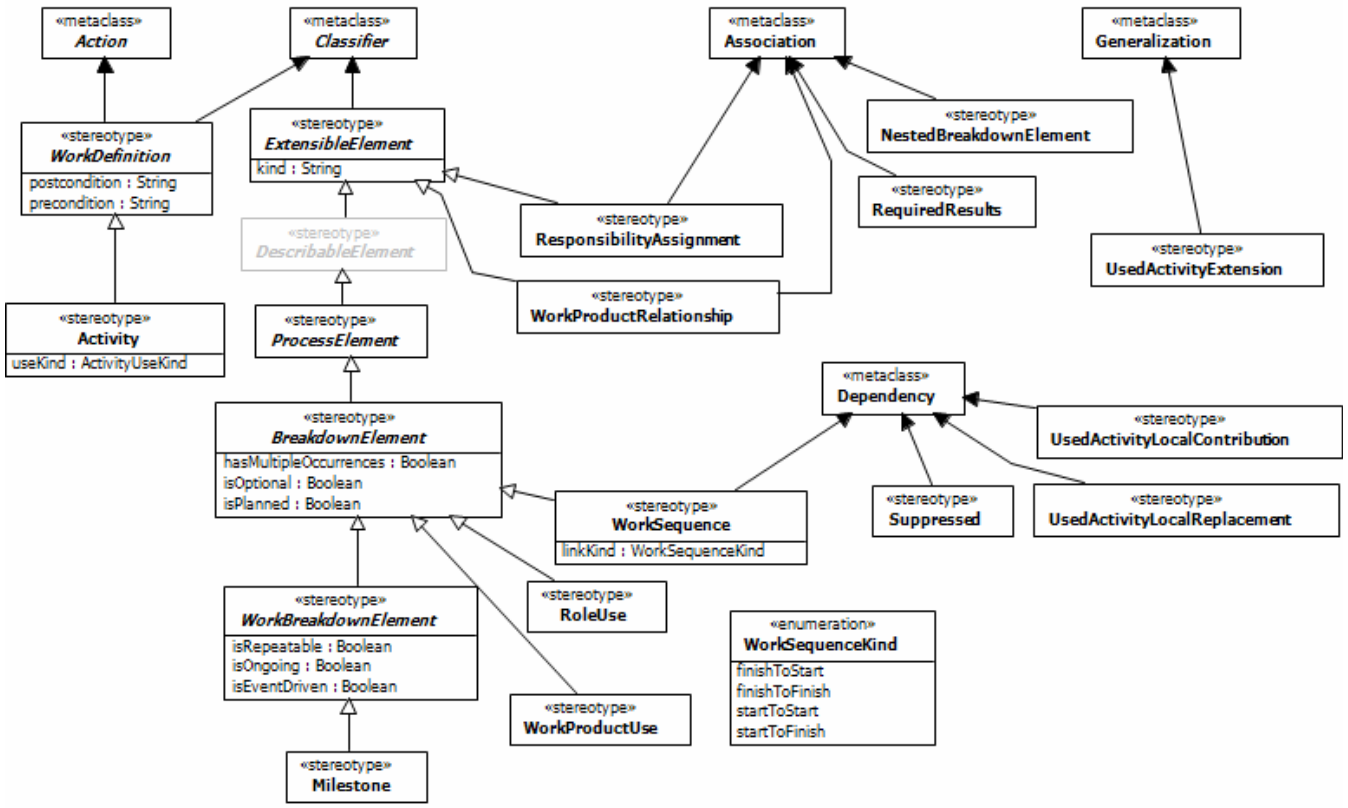


Figure 9.1 - The SPEM 2.0 UML 2 Profile stereotypes defined in the Process Structure package

In the SPEM 2.0 Meta-Model, processes are represented with a breakdown structure mechanism that defines a breakdown of Activities, which are comprised of other Activities or leaf Breakdown Elements such as Milestones or Role Uses. This breakdown structure mechanism is defined independent of the concrete lifecycle models the process engineer wants to express with them. In other words, the meta-model is able to represent 'different types of' processes, such as waterfall processes as well as iterative or incremental process models, by modeling them all as breakdown structures, but applying different structural relationships and descriptive attributes expressing their lifecycle specifics.

Moreover, the generic SPEM 2.0 meta-model mechanisms enable different types of presentations for breakdown structures, as illustrated with examples in the sections below. For instance, processes modeled as SPEM 2.0 breakdown structures can be presented as project plan-like structured list, workflow or activity diagrams, as well as static structure diagrams showing inputs/outputs to work as well as the performing roles.

Although a Process aims to describe the work to be performed in a development project, it keeps certain decisions that are too project specific open. For example, the breakdown structure defines which Breakdown Elements have multiple occurrences or is repeatable via its respective attributes, but does not say how many occurrences and how many repeats/ iterations it will have. These decisions have to be made by a project manager when planning a concrete project, project phase, or project iterations utilizing the attributes of Breakdown Element and Work Breakdown Element to identify the elements that need to be instantiated multiple times (allowing parallel execution) or repeatedly (being executed sequentially).

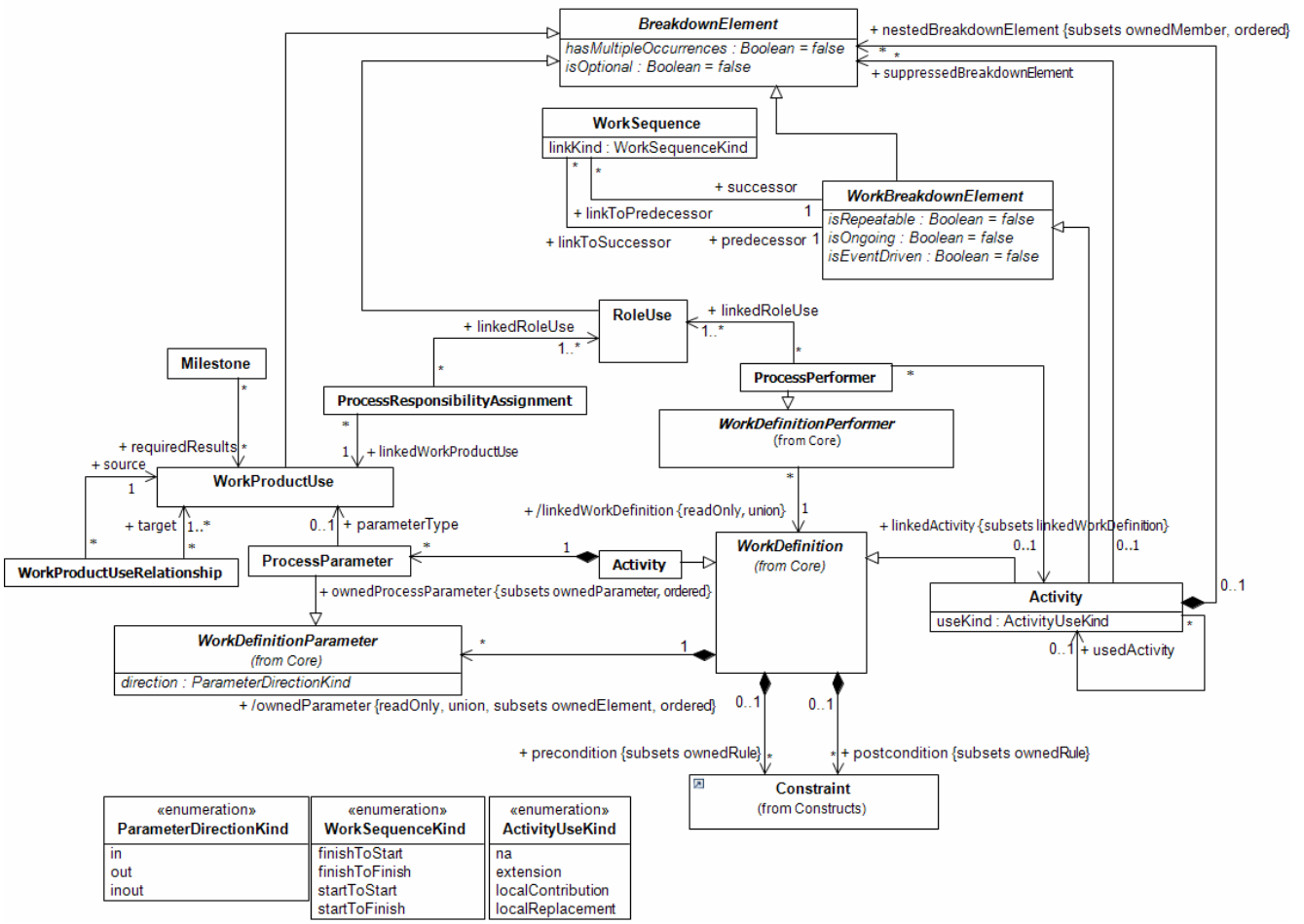


Figure 9.2 - Overview of the main classes and associations of Process Structure package (this diagram does not include all generalizations, see Figure 9-11 for the complete taxonomy)

Processes defined with this meta-model package's structures can represent ad-hoc processes that do not refer to any specific explicit methods to be applied within the process, but that do organize work as a breakdown of activities and role-work product responsibility allocation that typically rely on tacit knowledge for understanding what work such responsibilities entail. Popular examples for such processes are Scrum-based processes, which assume self-organizing teams, but define a general process for every sprint and across sprints defining key deliverables and milestones. Figure 9.3 depicts an example for such a process, which is organized into phases and iterations, as well as a breakdown of activities that is comprised of role use instances as well as work products each role use is responsible for. The activities

listed for an iteration in Figure 9.3 are high level and do not explicitly define what concrete work needs to be done on the work products listed (e.g., managing an iteration with an iteration plan and work items list can be done following many different management methods).

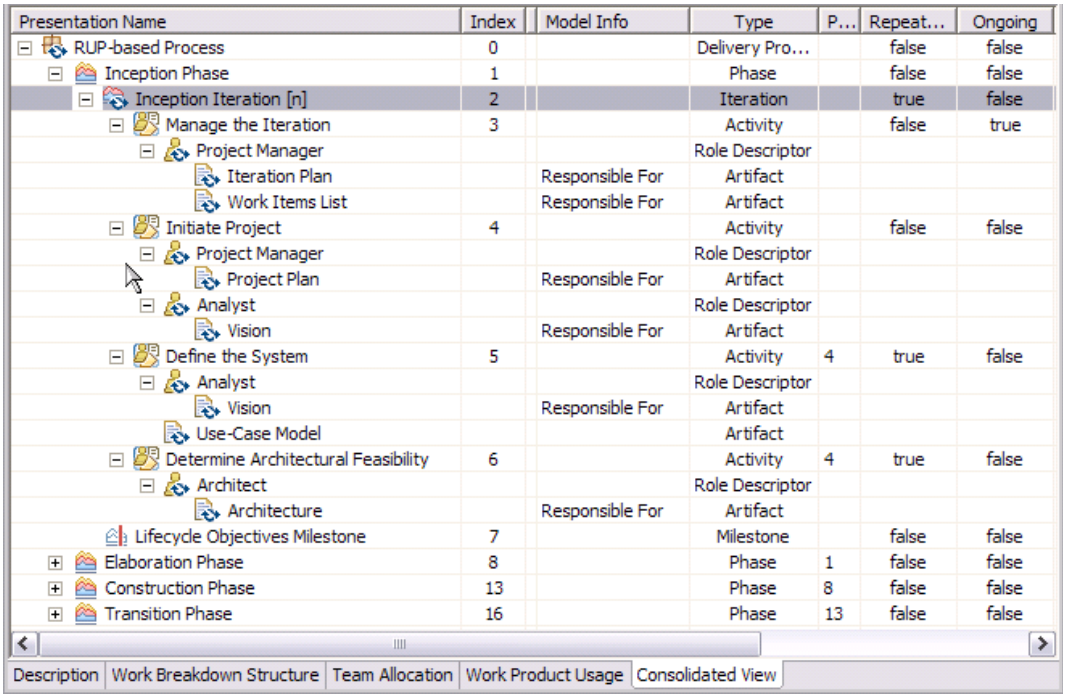


Figure 9.3 - Process Structure example: Multi-phased light-weight activity breakdown defining work product use responsibilities

Behavior of processes can be modeled by merging this package into Process Behavior (Chapter 10). Textual descriptions for development process structure elements and behavior can be added by merging this meta-model package into the Managed Content package (Chapter 11).

9.1 Activity

Super Class

Work Breakdown Element

Work Definition (from Core)

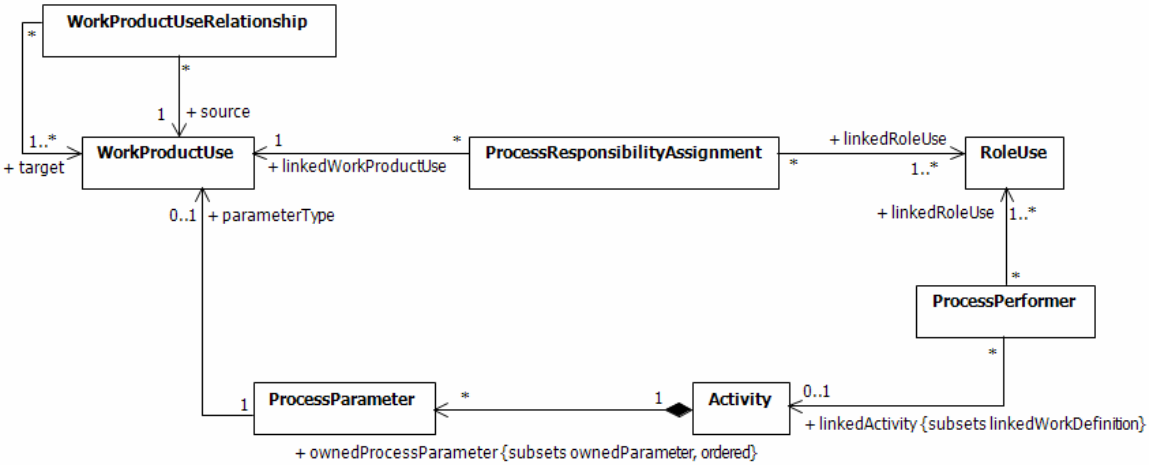


Figure 9.4 - The Work Definition Activity in relationship to its work performers and input/outputs

Description

An Activity is a Work Breakdown Element and Work Definition that defines basic units of work within a Process as well as a Process itself. In other words, every Activity represents a Process in SPEM 2.0. It relates to Work Product Use instances via instances of the Process Parameter class and Role Use instances via Process Performer instances.

Activity supports the nesting and logical grouping of related Breakdown Elements forming breakdown structures (not shown in Figure 9.4, see Figure 9.9). The concrete breakdown structure an Activity defines (i.e., its contained elements) can be reused by another Activity via the used Activity association (see Figure 9.5), which allows the second Activity to inherit its complete sub-structure (see Section 9.2 for details).

Attributes

- useKind: ActivityUseKind = na

  This attribute is set to a value of the enumeration Activity Use Kind other than its default ‘na’ when the Activity instance is associated to another Activity via the useActivity association for reuse. The value of the attribute of the Activity of the to-many end defines the semantics of the association according to the definitions in Section 9.2, Activity Use Kind.

Association Properties

- nestedBreakdownElement: BreakdownElement

  This association represents breakdown structure nesting. It defines an n-level hierarchy of Activities grouping together other Breakdown Elements such as other Activities, Milestones, etc. (see Figure 9.9)

• ownedProcessParameter: ProcessParameter	This composition association manages a list of ordered ProcessParameters for an Activity instance. The association property subsets the ownedWorkDefinitionParameter of Work Definition.
• suppressedBreakdownElement: Breakdown Element	The suppressed association allows hiding any Breakdown Element from the interpretation of a process structure. It is used in combination with the usedActivity association for elements that are inherited by usedActivity. The reusing activity can define its own local elements that refer to the base activity's element to denote the suppression (see Figure 9.5 and Section 9.2 for more details).
• usedActivity: Activity	This association defines a reuse generalization for Activities according to the semantics defined in Section 9.2, Activity Use Kind. Activity instances on the to-many end of the association can reuse properties of the Activity instance on the to-one end (base) of the association.

Semantics

Activity is a concrete Work Definition that represents a general unit of work assignable to specific performers represented by Role Use. An Activity can rely on inputs and produce outputs represented by Work Product Uses. Activity also represents a grouping element for other Breakdown Elements such as Activities, Method Content Uses, Milestones, etc. It is not per-se a 'high-level' grouping of only work as in other meta-models, but groups any kind of Breakdown Elements. For example, one can define valid Activities that group only Work Product Uses without any matching Role Uses or Parameters. Such a structure expresses the information that the Activity requires work on these Work Products, without specifying how and who, thus providing the flexibility required for modeling partial or Agile processes. Activity instances can inherit, contribute, or replace properties from other reused activities. The semantics for the three kinds of reuse are specified in 9.2, 'Activity Use Kind.'

SPEM 2.0 Profile Notation

The Activity stereotype is a subclass of the Work Definition stereotype. Actions can also be assigned the Activity stereotype as SPEM 2.0 uses Activity Use instead of Actions.

Stereotype	Meta-/ Superclass	Keyword	Properties	Abstract	Icon
Activity	WorkDefinition, Action	activity	n/a	no	SPEM 1.1 backwards compatibility icon:
NestedBreakdownElement	Association	nesting	n/a	no	n/a
UsedActivityExtension	Generalization	activity extension	n/a	no	n/a
UsedActivityLocalContribution	Dependency	local contribution	n/a	no	n/a
UsedActivityLocalReplacement	Dependency	local replacement	n/a	no	n/a
Suppressed	Dependency	suppressed	n/a	no	n/a

9.2 Activity Use Kind

Super Class

n/a : Enumeration

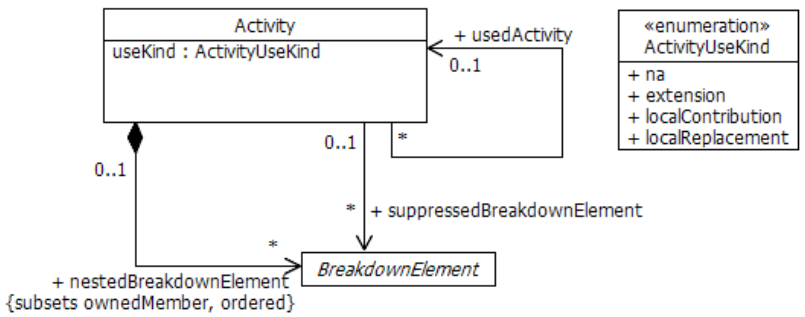


Figure 9.5 - Activity Use and Activity Use Kinds

Description

This enumeration defines the nature of the reuse for an Activity that relates to exactly one other Activity via the used Activity association. Activity Use in SPEM defines the ability to reuse the structures defined for one Activity via its nested Breakdown Element composition in a second Activity without the need to physically copy these structures. Instead, Activity Use defines a way to dynamically inherit these structures from the referenced the activity. Such reuse is typically established via an Extension relationship, and then further refined using additional local Contribution and local Replacement relationships among substructure elements of the extending activities.

Enumeration Literals

- na — This is the default value for Activities that do not instantiate the usedActivity association.
- extension — Extends provides a mechanism for dynamically reusing Activity substructures (elements contained via the nested Breakdown Element composition) in other Activities. Typical applications of Extension are to represent reusable process patterns as activities, which are then dynamically bound to different activities via the Extension use Kind in a larger process. An Activity is linked via extension by defining a usedActivity association instance from an Activity within the Processes to the Activity representing the process pattern and setting the useKind attribute to the Extension value. The source Activity inherits all association instances and sub-structures from the base Activity, and the Activity appears to be part of the resulting Process after interpretation of the used Activity association.
- localContribution — Local Contribution defines a mechanism for defining specific local additions (or contributions) to breakdown elements inherited via the extension Activity Use Kind within the context of the reusing Activity. Hence, usedActivity relationships of this kind are used in addition to a usedActivity relationship of the kind Extension. For example, an Activity could be defined as a child of a second Activity which defines an extension relationship to a third Activity adding additional new sub-elements (such as new sub-Activities) to the set of sub-elements the second Activity inherited from the third Activity via the extension relationship.
- localReplacement — Local Replace defines a mechanism for defining local replacements to specific breakdown elements inherited via the Extension Activity Use Kind in the context of the reusing Activity. UsedActivity relationships of this kind are used in addition to a usedActivity relationship of the kind Extension. Instances of this type replace inherited Activity sub-elements with their own sub-elements. For example, an Activity could be defined as a child of a second Activity which defines an extension relationship to a third Activity, replacing with its own set of sub-elements the inherited sub-elements (such as its sub-Activities) that the second Activity inherited from the third Activity via the extension relationship.

Semantics

The Extension Activity Use Kind provides mechanisms for dynamically linking Activities for reuse to other Activities or Processes as shown in the following example depicted in Figure 9.6.

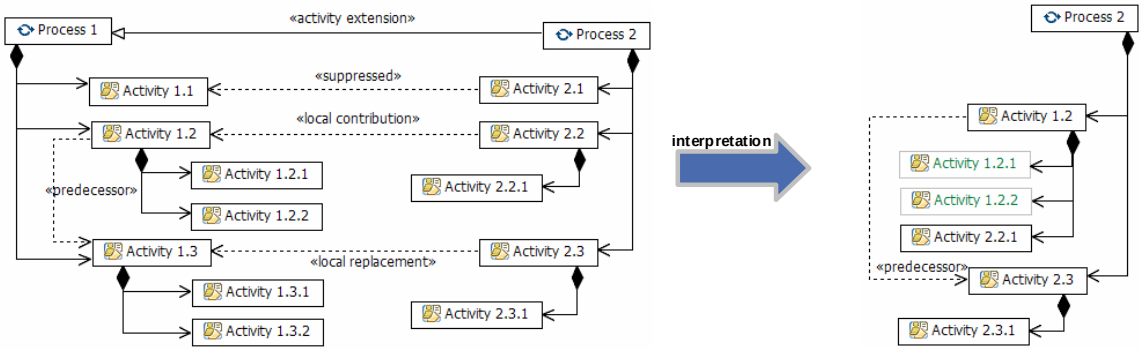
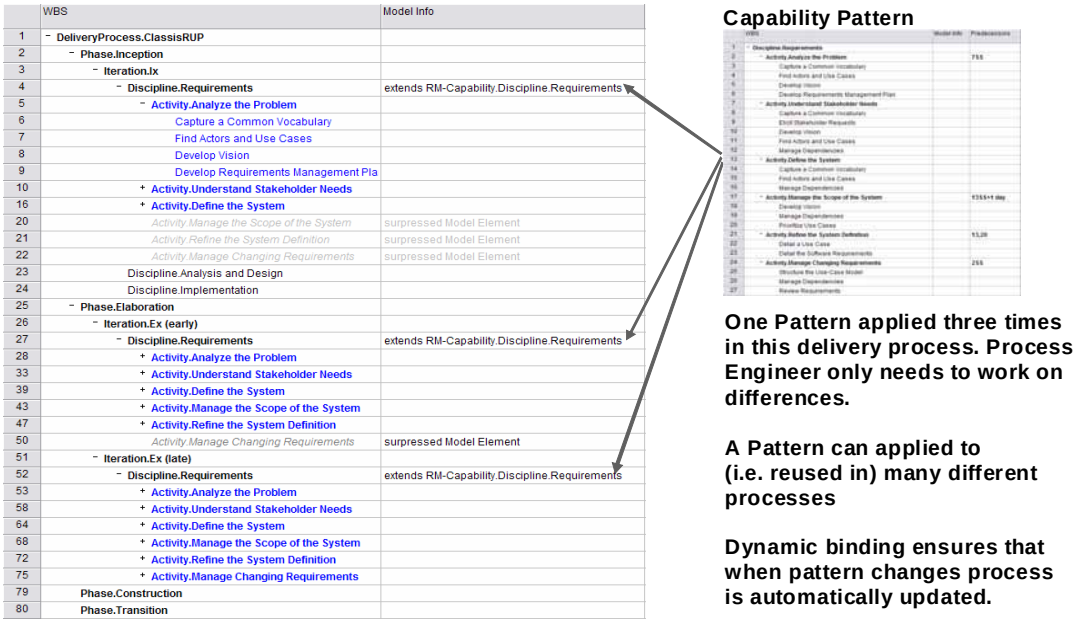


Figure 9.6 - Example of an activity reusing another activity

Figure 9.6 shows two processes modeled as activities with the SPEM 2 UML 2 Profile. Process 1 comprises of a breakdown of Activities. Process 2 reuses the structure of Process 1 via an activity extension through which Process 2 inherits the complete physical structure of Process 1. Hence, the interpretation of the activity extension relationship results in Process 2 comprising of all the Activity 1.x activities that Process 1 comprises of. However, the author who created Process 2 decided that he does not need all of the elements inherited from Process 1. He uses a suppressed relationship between a local activity that he created named Activity 2.1 and Activity 1.1 to indicate that he suppresses this activity from being inherited. Further, he models a local contribution with which he adds Activity 2.2.1 as an additional child to Activity 1.2. Finally, he creates a local replacement with Activity 2.3 replacing Activity 1.3 with a complete new set of children. The right side of Figure 9.6 shows the resulting interpretation of Process 2 with all the relationships modeled to Process 1. Note that the incoming predecessor relationship that pointed to Activity 1.3 now points to the replacing Activity 2.3.



	WBS	Model Info
1	DeliveryProcess.ClassisRUP	
2	Phase.Inception	
3	Iteration.Ix	
4	Discipline.Requirements	extends RM-Capability.Discipline.Requirements
5	Activity.Analyze the Problem	
6	Capture a Common Vocabulary	
7	Find Actors and Use Cases	
8	Develop Vision	
9	Develop Requirements Management Pla	
10	Activity.Understand Stakeholder Needs	
16	Activity.Define the System	
20	*Activity.Manage the Scope of the System*	surpressed Model Element
21	*Activity.Refine the System Definition*	surpressed Model Element
22	*Activity.Manage Changing Requirements*	surpressed Model Element
23	Discipline.Analysis and Design	
24	Discipline.Implementation	
25	Phase.Elaboration	
26	Iteration.Ex (early)	
27	Discipline.Requirements	extends RM-Capability.Discipline.Requirements
28	Activity.Analyze the Problem	
33	Activity.Understand Stakeholder Needs	
39	Activity.Define the System	
43	Activity.Manage the Scope of the System	
47	Activity.Refine the System Definition	
50	*Activity.Manage Changing Requirements*	surpressed Model Element
51	Iteration.Ex (late)	
52	Discipline.Requirements	extends RM-Capability.Discipline.Requirements
53	Activity.Analyze the Problem	
58	Activity.Understand Stakeholder Needs	
64	Activity.Define the System	
68	Activity.Manage the Scope of the System	
72	Activity.Refine the System Definition	
75	Activity.Manage Changing Requirements	
79	Phase.Construction	
80	Phase.Transition	

Figure 9.7 - Using Extends to dynamically bind a Process Pattern to a Delivery Process

The example in Figure 9.7 shows a Process Pattern (which is an example for a typical Kind (Section 8.2) for an Activity (Section 9.1)) on the right-hand side that defines a standard way of doing Requirements Management. The left-hand side shows a Delivery Process (another Process Kind) that is dynamically linking - one can also say applying - this Process Pattern via the Extension Activity Use Kind three times in three different locations (assuming that requirements work is iterated three times). These so-called pattern applications have been created not by copying the breakdown structure of the Process Pattern into the breakdown structure of the Delivery Process, but by creating an Extension Activity Use association from an Activity in the Delivery Process to the top-level Activity in the Process Pattern. Through the interpretation of the Extension relationship, all sub-Activities and Method Content Uses typed in blue in Figure 9.7 have been applied through value inheritance and appear in the Delivery Process. In addition to just applying the pattern as is, the example also shows that for every application, specific modifications have been specified. Using the Process Element attribute 'suppressed,' elements can be suppressed from the pattern application.

Figure 9.7 shows that in the first pattern application, three Activities have been suppressed (indicated by gray color as well as the text in the Model Info column). Suppression means that these Activities will not be performed in that particular application of the pattern. In the second application of the pattern, one can see that fewer, and in the third application no, activities have been suppressed. This is because in these instances of the pattern, these activities shall be performed. The true power of using Extension for applying Process Patterns to a Process is the increase in maintainability. We see in this small example that the pattern has been applied three times. In a real Delivery Process definition they might be applied many more times. If a process engineer now wants to make a change to the Process Pattern, s/he only has to make this change once in the original pattern definition. Through the dynamic binding and interpretation, all changes will be automatically visible in all pattern application. In cases where s/he does not want those changes to be propagated to specific pattern application, s/he could create a physical deep copy of the pattern and break the Extension link.

Figure 9.8 shows another view of the same Delivery Process, which does not yet apply the interpretation of the Extension relationship. Whereas Figure 9.7 presented the results of fully interpreting the Extension relationships, this example shows what actually gets modeled to achieve this result. One can see that the Activity "Discipline.Requirements" defines the extension to the Process Pattern. All sub-elements of that Activity will be inherited from that base Activity when the relationship is interpreted. Only the elements to be suppressed have been modeled and added to the Activity as nested elements to set their suppression attribute. By matching the name, the interpretation will suppress these Activities with all their nested sub-elements from the Extends interpretation.

	Name	Model Info	Predecessors
1	**Phase.Inception**		
2	**Iteration.Ix**		
3	**Discipline.Requirements**	extends RM-Capability.Discipline.Requirements	
4	*Activity.Manage the Scope of the System*	surpressed Model Element	
5	*Activity.Refine the System Definition*	surpressed Model Element	
6	*Activity.Manage Changing Requirements*	surpressed Model Element	
7	Discipline.Analysis and Design		
8	Discipline.Implementation		
9	**Phase.Elaboration**		
10	**Iteration.Ex (early)**		
11	**Discipline.Requirements**	extends RM-Capability.Discipline.Requirements	
12	*Activity.Manage Changing Requirements*	*surpressed Model Element*	
13	**Iteration.Ex (late)**		
14	**Discipline.Requirements**	extends RM-Capability.Discipline.Requirements	
15	Phase.Construction		
16	Phase.Transition		

Figure 9.8 - Representation of the Delivery Process from Figure 9.7 before interpretation of the Extends relationship

Local Contribution allows adding process elements into a process structure inherited via a second Extension Activity Use relationship. It allows updating a Process Pattern or activity that is dynamically linked into a process a-posteriori with new capabilities. Local Replacement allows replacing process structures inherited via Extension Activity Use relationship with new breakdown elements.

	WBS	Model Info	Predecessor
1	- **ProcessContribution.Creativity Based Requirements Elicitation**		
2	- **Discipline.Requirements**		
3	- **Activity.Analyze the Problem**	**replaces ClassicRUP.Inception.Ix.Requirements.Analyze the Problem**	
4	Goal Definition Workshop		
5	Contextual Inquiry		
6	Creative brainstorming session		
7	- **Activity.Understand Stakeholder Needs**	**contributes to ClassicRUP.Inception.Ix.Requirements.Understand Stakeholder Needs**	
8	Goal Operationalization Workshop		ClassicRUP.Inception.Ix.Requirements.Understand Stakeholder Needs.ElicitStakeholderRequests

Figure 9.9 - Modeling Local Contribution and Local Replacement

Figure 9.9 and Figure 9.10 depict examples for Local Contribution and Local Replacement. They show two Activities that each refers to Activities in the Delivery Process depicted earlier in Figure 9.7. Figure 9.10 shows the resulting process after interpretation of the activity use relationships. The first Activity defines a replacing Activity that relates to the Inception Phase's "Analyze the Problems" Activity. It replaces this Activity with a complete new breakdown structure. Figure 9.9 only depicts the Work Breakdown Structure view showing the Activities new sub-Activities, but a Local Replacement could provide updates starting at any level of the breakdown. The second Activity in Figure 9.9 defines a Local Contribution Activity that adds to the Delivery Process' "Understand Stakeholder Needs" Activity in Inception an additional Activity "Goal Operationalization Workshop." It also defines a predecessor for this contributed new Activity in the Delivery Process.

	WBS	Model Info	Predecessors
1	- DeliveryProcess.ClassisRUP		
2	- Phase.Inception		
3	- Iteration.Ix		
4	- Discipline.Requirements	extends RM-Capability.Discipline.Requirements	
5	- Activity.Analyze the Problem	replaced by Creativity Based Requirements Elicitation.Requirements.Analyze the Problem	
6	Goal Definition Workshop	from Creativity Based Requirements Elicitation.Requirements.Analyze the Problem	
7	Contextual Inquiry	from Creativity Based Requirements Elicitation.Requirements.Analyze the Problem	
8	Creative brainstorming session	from Creativity Based Requirements Elicitation.Requirements.Analyze the Problem	
9	- Activity.Understand Stakeholder Needs	contributed to by Creativity Based Requirements Elicitation.Requirements.Understand Stak	
10	Capture a Common Vocabulary		
11	Elicit Stakeholder Requests		
12	Goal Operationalization Workshop	from Creativity Based Requirements Elicitation.Requirements.Understand Stakeholder Needs	11
13	Develop Vision		
14	Find Actors and Use Cases		
15	Manage Dependencies		
16	* Activity.Define the System		
20	*Activity.Manage the Scope of the System*	surpressed Model Element	
21	*Activity.Refine the System Definition*	surpressed Model Element	
22	*Activity.Manage Changing Requirements*	surpressed Model Element	
23	Discipline.Analysis and Design		
24	Discipline.Implementation		
25	- Phase.Elaboration		
26	- Iteration.Ex (early)		
27	* Discipline.Requirements	extends RM-Capability.Discipline.Requirements	
51	- Iteration.Ex (late)		
52	* Discipline.Requirements	extends RM-Capability.Discipline.Requirements	
79	Phase.Construction		
80	Phase.Transition		

Figure 9.10 - The resulting Delivery Process after interpreting the Variability Specializations

The resulting Process after interpreting the Activity Use relationships for the Delivery Process is depicted in Figure 9.10. The blue entries show the Breakdown Elements that arrive via the Local Replacement and Local Contribution interpretation.

SPEM 2.0 Profile Notation

See Activity (Section 9.1)

9.3 Breakdown Element

Super Class

Process Element

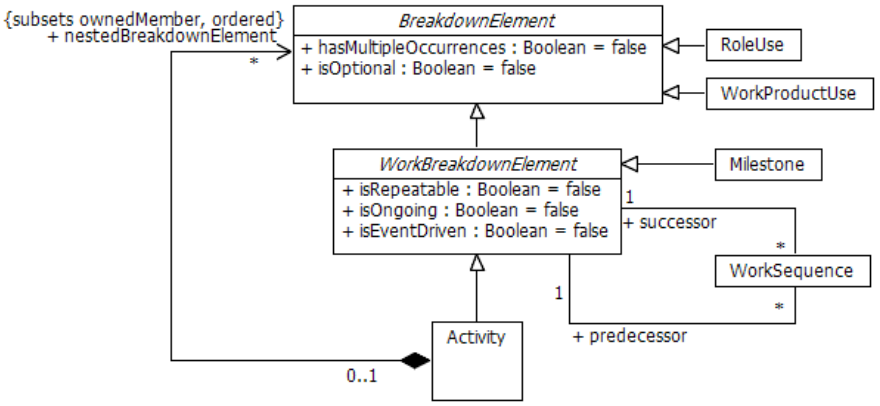


Figure 9.11 - A breakdown structure is defined by Activities nesting Breakdown Elements

Description

Breakdown Element is an abstract generalization for any type of Process Element that is part of a breakdown structure. It defines a set of properties available to all of its specializations. Any of its concrete subclasses can be 'placed inside' an Activity (via the nested Breakdown Element association) to become part of a breakdown of Activities as well as the Activities namespace. As Activities are Breakdown Elements themselves and therefore can be nested inside other activities, an n-level breakdown structure is defined by n nested Activities. In addition to Activities, other Breakdown Elements as shown in Figure 9.11 can be nested inside Activities as leaf elements of the breakdown.

Attributes

- hasMultipleOccurrences: Boolean = false

  The hasMultipleOccurrences attribute set to 'true' expresses that when the process is enacted, the respective Breakdown Element will typically occur in more than one instance (i.e., more than one instance for the Breakdown Element instance).
  This might provide important guidance for creating plans from a Process. For example, a Work Definition such as "Detail Use Case" would be performed for every use case identified for a particular Iteration or Activity. Generating a plan would list one Work Definition instance per use case, each listing a different use case instance as the input/output. In contrast to the isRepeatable attribute defined for Work Breakdown Element (Section 9.10) the hasMultipleOccurrences attribute does not assume any dependencies among the occurrences of the Breakdown Element. For example, the "Detail Use Case" Work Definition occurrences mentioned above can be performed in parallel as well as one after the other.

- isOptional: Boolean = false

The isOptional attribute indicates that the Breakdown Element describes work, a work product, or even work performer, which inclusion is not mandatory when performing a project that is planned based on a process containing this element.

Semantics

No additional semantics. See subclasses.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
BreakdownElement	ProcessElement	n/a	hasMultipleOccurences, isOptional	yes	n/a

9.4 Milestone

Super Class

Work Breakdown Element

Description

A Milestone is a Work Breakdown Element that represents a significant event for a development project.

Association Properties

- requiredResults: WorkProductUse

This association links the Work Product Uses instances to a Milestone instance that need to be produced for that Milestone.

Semantics

A Milestone describes a significant event in a development project, such as a major decision, completion of a deliverable, or meeting of a major dependency (like completion of a project phase). Because Milestone is commonly used to refer to both the event itself and the point in time at which the event is scheduled to happen, it is modeled as a Work Breakdown Element (i.e., it appears as part of a breakdown structure and can have predecessors).

SPEM 2.0 Profile Notation

Stereotype	Superclass/Metaclass	Keyword	Properties	Abstract	Icon
Milestone	WorkBreakdownElement / Classifier	milestone	n/a	no	
RequiredResults	Association	required results	n/a	no	n/a

9.5 Process Element

Super Class

Extensible Element (Core)

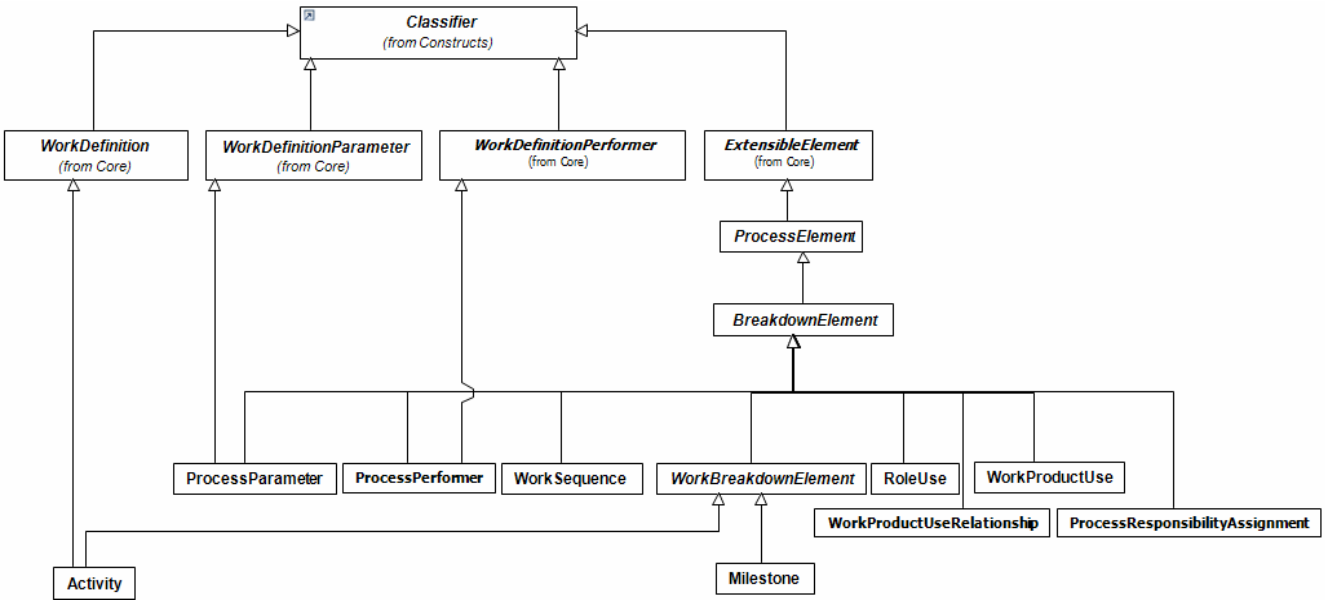


Figure 9.12 - Taxonomy of classes defined in Process Structure package

Description

Process Elements is an Extensible Element that represents abstract generalization for all elements that are part of a SPEM 2.0 Process.

Semantics

A Process Element is any element that is part of a Process. See Figure 9.12 for a complete taxonomy of all Process Elements defined in the Process Structure meta-model package.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
ProcessElement	BreakdownElement	n/a	n/a	yes	n/a

9.6 Process Parameter

Super Class

Work Definition Parameter

Breakdown Element

Description

A Process Parameter is a Work Definition Parameter and Breakdown Element that is used for process definitions. It defines input and output meta-types to be Work Product Uses.

Association Properties

- parameterType: WorkProductUse — This association links zero or one Work Product Use instances to a parameter. Processes could leave the type specification open and not specify a concrete Work Product Use.

Semantics

No additional semantics defined.

SPEM 2.0 Profile Notation

Use Work Definition Parameter (see Section 8.5).

9.7 Process Performer

9.7.1 Super Class

Breakdown Element

Work Definition Performer (from Core)

Description

A Process Performer is a Breakdown Element and Work Definition Performer that represents a relationship between Activity instances and Role Use instances. An instance of Process Performer links one or more Role Use instances to one Activity. (Modeled as 0..1 Activities because the Process with Methods meta-model package will add an alternative class to Activity.)

Association Properties

- linkedActivity: Activity — A Process Performer links to zero or one Activity. The linked Activity property subsets the linked Work Definition property from the Work Definition Performer defined in Core.
- linkedRoleUse: RoleUse — A Process Performer links to one or more Role Use.

Semantics

The Process Performer links Role Uses to Activities, indicating that these Role Use instances participate in the work defined by the activity in one or another way. The kind of involvement of the Role Use in the Activity needs to be defined by Kind (Section 8.2) class instances that qualify the Process Performer instances. Typical examples for Kinds of Process Performers would be Primary Performer, Additional Performer, Assisting Performer, Supervising Performer, Consulted Performer, etc. The popular RACI-VS diagram defines another set of commonly used Kinds for the Process Performer: Responsible, Accountable, Consulted, Informed, Verifies, and Signs.

SPEM 2.0 Profile Notation

Use Work Definition Performer (see Section 8.6).

9.8 Process Responsibility Assignment

Super Class

Breakdown Element

Description

A Process Responsibility Assignment is a Breakdown Element that represents a relationship between instances of Role Use and Work Product Use. An instance of the Process Responsibility Assignment links one or more Role Use instances to exactly one Work Product Use.

Association Properties

- linkedRoleUse: RoleUse — A Process Responsibility Assignment links to one or more Role Use.
- linkedWorkProductUse: WorkProductUse — A Process Responsibility Assignment links to exactly one Work Product Use.

Semantics

The Process Responsibility Assignment links Role Uses to Work Product Uses indicating that the Role Use has a responsibility relationship with the Work Product Use. The Kind of responsibility of the Role Use for the Work Product Use needs to be defined by Kind (Section 8.2) class instances that qualify the Process Responsibility Assignment. The popular RACI-VS diagram defines a set of commonly used Kinds which cannot only be applied for the Process Performer, but are also often used for work product responsibility: Responsible, Accountable, Consulted, Informed, Verifies, and Signs.

Note that for many processes the Process Performer and Process Responsibility represent two quite different sets of information as a Role Use can be involved in an Activity that modifies a work product without being responsible for the Work Product itself and vice versa: A Role Use can be responsible for a Work Product Use without participating in all the Activities that modify it. Other processes might choose to only use one of these relationships. For example, there are processes for self-organizing teams that do not define detailed Activities describing which Role Use performs which work. These approaches might just utilize the Process Responsibility Assignment to express that a Role Use has a certain responsibility, but that the process does not prescribe how that responsibility is achieved. In such a case the Kinds defined might express work product states such as ‘detailed,’ ‘implemented,’ ‘documented,’ ‘reviewed,’ and so on.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
ResponsibilityAssignment	ExtensibleElement / Association	responsible	n/a	no	n/a

9.9 Role Use

Super Class

Breakdown Element

Description

A Role Use is a special Breakdown Element that either represents a performer of an Activity or a participant of the Activity. If it is a performer, the Role Use and Activity need to be related via a Process Performer. If it is a participant, then the Role Use is stored in the nestedBreakdownElement composition of the Activity and might be used by one of the sub-activities as a performer and/or a Process Responsibility Assignment. Role Uses are only valid within and specific to the context of an Activity. They are not to be reused across activities.

Semantics

A Role Use represents an activity-specific occurrence of an activity performer or participant. A Role Use is an activity-specific object and not a general reusable definition of an organizational role. (The meta-model package Method Content defines a concepts call Role Definition for that in Section 12.7). A Role Use represents the occurrence of a real person performing activity-specific work and having activity-specific responsibilities. A Role Use participant stored with an Activity can be only accessed and reused by the Activity’s sub-Activities and not by any parent or sibling Activities in the Activity breakdown structure. This scoping of Role Use in the local namespace of Activities allows different Performers as well as different Responsibility Assignments for every Activity. This has been done to reflect the fact that work product responsibilities and performance of activities might change throughout the development lifecycle. In other words, Role Use instances with the same name, but different responsibilities and performing different work, can be created in different Activities.

Many processes do not comprise of Role Definitions and just define the occurrence of a role and imply that all team members know what the Role Use represents (e.g., by just interpreting the Role Use name). In these situations, fitting individuals for the Role Uses are assigned when the process is enacted. Assigning Role Uses in such a way is also often referred to as ‘self-organizing’ teams in which team members choose by themselves or via their managers which roles they play during development.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
RoleUse	BreakdownElement / Classifier	role use	n/a	no	SPEM 1.1 backwards compatibility icon:

9.10 Work Breakdown Element

Super Class

Breakdown Element

Description

A Work Breakdown Element is a special Breakdown Element that provides specific properties for Breakdown Elements that represent work (see Figure 9.11). The properties are specific to breakdown structures and do not apply to all Work Definition subclasses.

Attributes

- isRepeatable: Boolean = false

  This attribute is used to define repetition of work, e.g., iterations. A Work Breakdown Element with this attribute set to True shall be repeated more than once on the same set of artifacts. For example, if one wants to model that a specific Activity instance shall represent an iteration in a process, then this attribute shall be set to true indicating that the Activity and therefore all of its sub-Activities will be performed more than once.
  The difference to the hasMultipleOccurrences attribute defined for Breakdown Element is that Work Breakdown Elements with the isRepeatable flag will be performed one after the other (i.e, not in parallel). For Breakdown Elements with hasMultipleOccurrences set to true this is undefined and up to the project planner. If both attributes are set to true, then isRepeatable takes precedence.

- isOngoing: Boolean = false

  If the isOngoing attribute is set to true for a Work Breakdown Element instance, then the element describes an ongoing piece of work without a fixed duration or end state. For example, the Work Breakdown Element could represent work of an administrator continuously (e.g., 3h a day) working to ensure that systems are kept in a certain state. Another example would be program management work overseeing many different projects being scheduled for one particular project at specific reoccurring intervals during the whole lifecycle of the project.

- isEventDriven: Boolean = false

  The isEventDriven attribute indicates that the Work Breakdown Element describes an instance of work that is not started because it has been scheduled to start at a certain point of time, preceding work is being completed, or input work products are available, but because another specific event has occurred.
  Examples for such events are exceptions or problem situations that require specific work to be performed as a result. Also, change management work can be modeled as event driven work analyzing a change request or defect and allocating work dynamically to resources to deal with it following the work described with such Work Breakdown Element. The events themselves are not modeled in this version of the specification. They shall be described as part of the normal descriptions fields available.

Association Properties

- linkToPredecessor: WorkSequence

  This association links a Work Breakdown Element to its predecessor. Every Work Breakdown Element can have predecessor information associated to it. This predecessor information is stored in instances of the class Work Sequence that defines the kind of predecessor another Work Breakdown Element represents for another.

- linkToSuccessor: WorkSequence

  This association links a Work Breakdown Element to its successor. Every Work Breakdown Element can have successor information associated to it. This successor information is stored in instances of the class Work Sequence that defines the kind of successor another Work Breakdown Element represents for another.

Semantics

Work Breakdown Element represents a work-specific breakdown element to be used in a work breakdown structure.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
WorkBreakdownElement	BreakdownElement	n/a	isReatable, isOngoing, isEventDriven	yes	n/a

9.11 Work Product Use

Super Class

Breakdown Element

Description

A Work Product Use is a special Breakdown Element that either represents an input and/or output type for an Activity or represents a general participant of the Activity. If it is an input/output, then the Work Product Use needs to be related to the Activity via the Process Parameter class. If it is a participant, then the Work Product Use is stored in the nestedBreakdownElement composition of the Activity and might be used by one of the sub-activities as an input/output and/or be related to a Role Use via a Process Responsibility Assignment. Role Use instances are only valid within and specific to the context of an Activity and not to be reused across activities.

Semantics

A Work Product Use represents an activity-specific occurrence of a Work Product input/output type or an Activity participant. A Work Product Use instance is an activity-specific object and not a general reusable definition of a work product. (The meta-model package Method Content defines a concepts call Work Product Definition for that in Section 12.5). A Work Product Use represents the occurrence of a real Work Product in the context of an activity. A Work Product Use participant stored with an Activity can only be accessed and reused by the Activity's sub-Activities and not by any parent or sibling Activities in the Activity breakdown structure. This scoping of Work Product Use in the local namespace of Activities allows the modeling of different Responsibility Assignments for every Activity, which reflects the fact that work product responsibilities might change throughout the development lifecycle. In other words, Role Use instances with the same name can be created in different Activities. All of these Role Use instances could all have different Work Product Use responsibilities.

Many processes do not comprise Work Product Definitions and just define the occurrence of a work product and imply that all team members know what the Work Product Use represents (e.g., by just interpreting the Work Product Use name). For example a process might define that Stakeholder Requests need to be captured, but does not formally define what a Stakeholder Request is nor prescribes any templates or forms for them. Team members could choose by themselves which form of representation they choose, as well as shift responsibilities for the Work Product during the lifecycle phases (i.e., Activities).

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
WorkProductUse	BreakdownElement / Classifier	work product use	n/a	no	

9.12 Work Product Use Relationship

Super Class

Breakdown Element

Description

A Work Product Use Relationship expresses a general relationship among work products. Kind (Section 8.2) class instances shall be used to specify the nature of this relationship.

Association Properties

• source: Work Product Use	This association links to the exact one source of the Work Product Use Relationship.
• target: Work Product Use	This association links to one or more targets of the Work Product Use Relationship.

Semantics

The Work Product Use Relationship can be used to express different kinds of relationships among Work Products Uses. Typical Kinds are ‘composition’ expressing that a work product use instance of an instance is part of another work product instance of an instance. For example, as depicted in Figure 9.13 (using the SPEM 2.0 UML 2 Profile in which the creator of the ‘composition’ Kind has decided to use the UML composition association), an instance of Actor is part of an instance of Use Case Model. In contrast to composition, another Kind could express ‘aggregation’ indicating that a Work Product Use is used with another Work Product Use. For example, a customer design deliverable could be defined as a compilation of other different work product uses that are assembled as a report that is delivered to the customer for review.

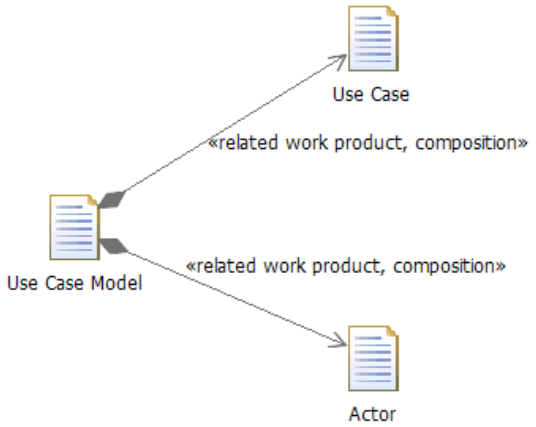


Figure 9.13 - Work Product composition example

A third commonly used Kind is ‘dependency’ or ‘impactedBy’ indicating that a work product use impacts another work product use. For example, if a use case model work product changes, then the use case realization work product needs to be updated with these changes. This dependency Work Product Use Relationship is used for work products diagrams as shown in Figure 9.15, specifying traceability dependencies among instances of work products to be considered when creating and changing these work products. Such dependencies are typically defined in a process for specific parts of a lifecycle, i.e., diagrams such as Figure 9.15 might be different for each project phase or even each Activity.

Work Product Uses represent work product references with unique attributes and associations within the scope of an Activity (including iterations or phases, which are modeled as Activities with these Kinds). A process engineer could model a phase for a process not only by defining Activities in a work breakdown structure, but also by first deciding which key work products shall be used, produced, and updated by the Activities of that phase. S/he would begin by producing a Work Product Dependency diagram as depicted below in Figure 9.14, continue by defining different states for each work product type as depicted in Figure 9.15, and then continue by allocating the achievement of states to specific Activities as depicted in Figure 9.16. (Note that Figure 9.14 and Figure 9.15 show to different presentation options: Whereas Figure 9.14 uses a proprietary arrow notation, Figure 9.15 uses SPEM 2.0 Profile stereotypes on a UML 2 class diagram based notation.)

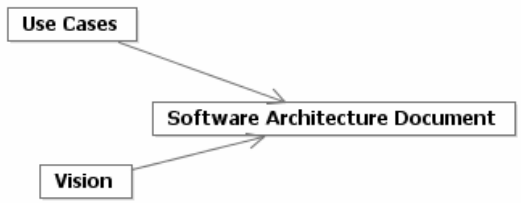


Figure 9.14 - Simple Work Product Dependency Diagram drawing impacts relations as arrows

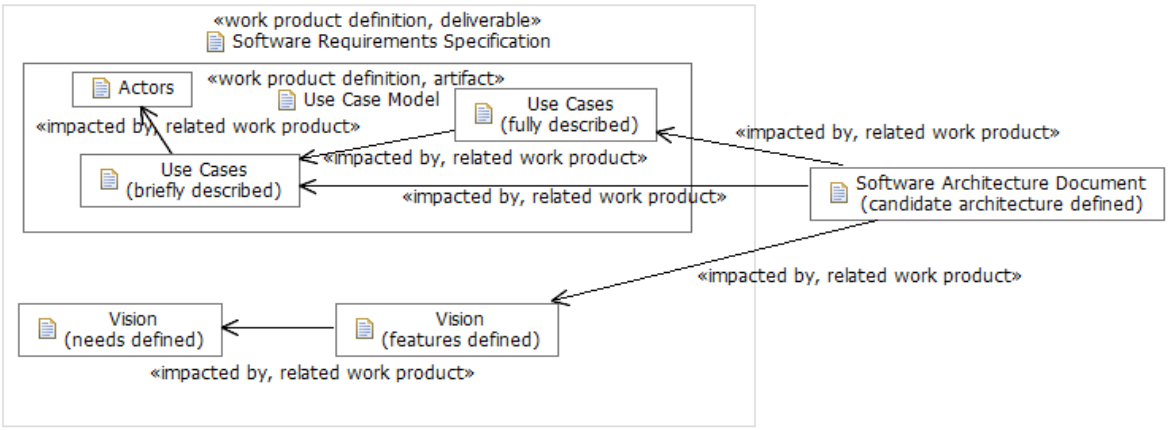


Figure 9.15 - Detailed UML 2-based Work Product Dependency diagram showing States as well Deliverable aggregation (Software Requirements Specification) and Work Product composition (Use Case Model)

The lower part of Figure 9.16 presents the outcome of such a procedure. It shows a Work Product Breakdown Structure (WPBS) of a process' Inception Phase in which Work Product Uses have been allocated to the Activities "Understand Stakeholder Needs" and "Define the System." Work Product Uses for the Artifacts "Glossary," "Vision" (one Method Content Use for each Activity), and "Use Case Model" have been created for each Activity. Moreover, Entry and Exit States have been defined for the Work Product Uses indicating the state change expected for each Work Product instance during each Activity. We see for example that the "Vision" document shall be in the state "Needs defined" after performing the Activity "Understand Stakeholder Needs" and in the state "Features defined" after the Activity "Define the System."

Once these states have been defined, sub-Activities (or alternatively Task Uses, which get introduced in the meta-model package Process with Methods in Section 13.14) that establish these states can be allocated to the Activities by instantiating respective sub-Activities (or Task Uses), completing the work breakdown as shown in the upper part of Figure 9.16. A CASE tool implementing SPEM 2.0 would be able to propose fitting Activities (or Task Uses) to the user for inclusion in this breakdown by searching available Activities (or Task Uses) that have these Work Product Uses as output. In this example, the result uses the concept of Task Uses (introduced in Section 13.14). You can see two Task Uses for "Develop Vision" in Figure 9.16. This Task itself as it would be defined as a Task Definition in Method Content. It would describe all aspects of developing a Vision document. However, for each of the shown Activities, a Task Use would point out only those parts of the Task that need to be performed to provide the specified state change. This way, processes define what parts of a method shall be performed at what point in time, providing the user with a unique focus on the work at hand by proving a selection of the complete method description.

Work Breakdown Structure and Work Product Breakdown Structure provide alternative but consistent views on a process as depicted in Figure 9.16. As described above, the picture shows the same process from the Work Breakdown Structure view showing the breakdown of Activities with Task Uses as well as from the Work Product Breakdown Structure view showing participating Work Products under Activities. The later view can be further refined by distinguishing Work Products Uses that are used as input versus output (or both) within the Activity.

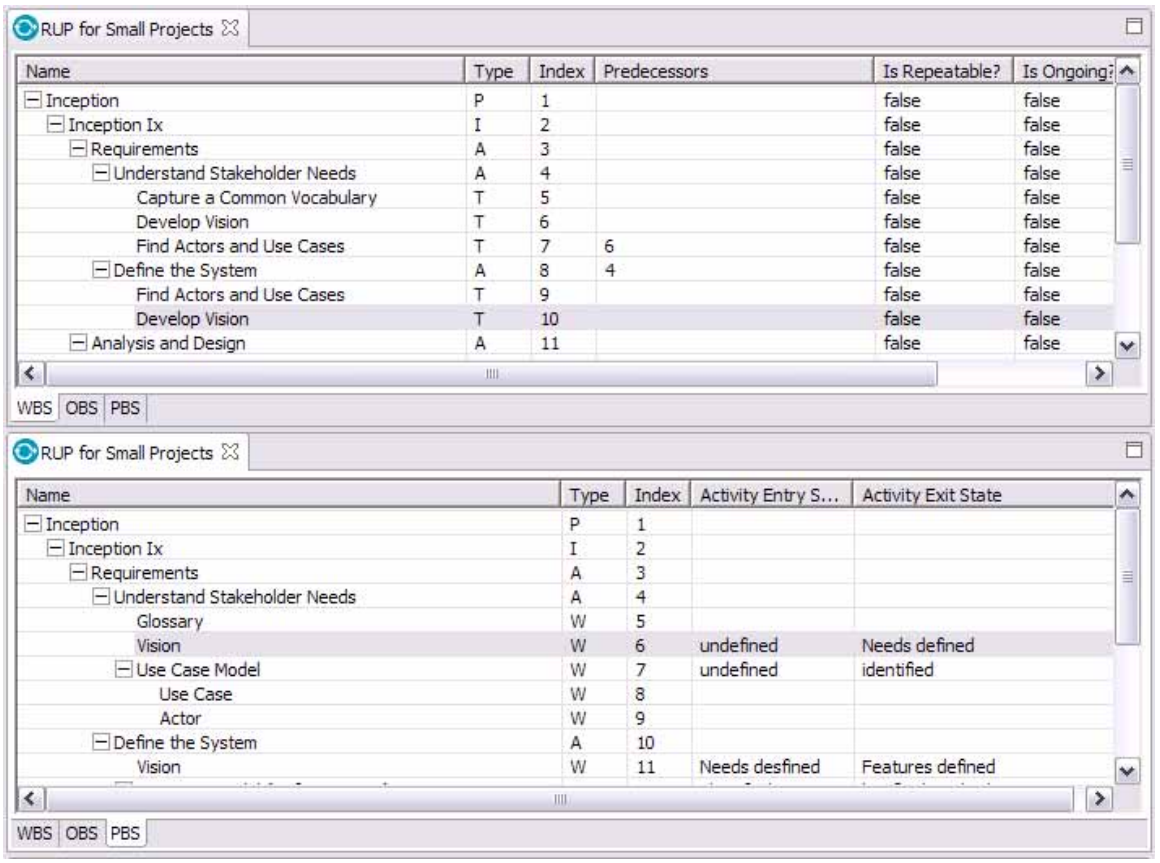


Figure 9.16 - Two views on the same process showing a Work Breakdown Structure as well as Work Product Breakdown Structure

SPEM 2.0 Profile Notation

SPEM 2.0 user can define their own stereotypes for the Kind classes that they produce or use the Kind property inherited from the Extensible Element stereotype. The Base Plugin defines a set of typical Kinds and stereotypes in Section 18.5.

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
WorkProductRelationship	ExtensibleElement / Association	related work product	n/a	no	n/a

9.13 Work Sequence

Super Class

Breakdown Element

Description

Work Sequence is a Breakdown Element that represents a relationship between two Work Breakdown Elements in which one Work Breakdown Elements depends on the start or finish of another Work Breakdown Elements in order to begin or end (see Figure 9.2).

Attributes

- linkKind: WorkSequenceKind — This attribute expresses the type of the Work Sequence relationship by assigning a value from the Work Sequence Kind enumeration.

Association Properties

- successor: WorkBreakdownElement — This association links a Work Breakdown Element to its successor. Every Work Breakdown Element can have successor information associated to it. This successor information is stored in instances of the class Work Sequence that defines the kind of successor another Work Breakdown Element represents for another.
- predecessor: WorkBreakdownElement — This association links a Work Breakdown Element to its predecessor. Every Work Breakdown Element can have predecessor information associated to it. This predecessor information is stored in instances of the class Work Sequence that defines the kind of predecessor another Work Breakdown Element represents for another.

Semantics

The Work Sequence class defines predecessor and successor relations among Work Breakdown Elements. This information is in particular critical for use of the process in planning applications. See more details on the different kinds of Work Sequence relationships in Section 9.14.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
WorkSequence	BreakdownElement / Dependency	predecessor	linkKind	no	n/a

9.14 Work Sequence Kind

Super Class

n/a : Enumeration

Description

Work Sequence represents a relationship between two Work Breakdown Elements in which one Work Breakdown Element (referred to as (B) below) depends on the start or finish of another Work Breakdown Element (referred to as (A) below) in order to begin or end. This enumeration defines the different kinds of Work Sequence relationships available in SPEM 2.0 and is used to provide values for Work Order's linkKind attribute.

Enumeration Literals

- finishToStart — Work Breakdown Element (B) cannot start until Work Breakdown Element (A) finishes. For example, if you have two Work Breakdown Elements, "Construct fence" and "Paint fence," "Paint fence" can't start until "Construct fence" finishes. This is the most common type of dependency and the default for a new Work Sequence instance.
- finishToFinish — Breakdown Element (B) cannot finish until Work Breakdown Element (A) finishes. For example, if you have two Work Breakdown Elements, "Add wiring" and "Inspect electrical," "Inspect electrical" can't finish until "Add wiring" finishes.
- startToStart — Breakdown Element (B) cannot start until Work Breakdown Element (A) starts. For example, if you have two Work Breakdown Elements, "Pour foundation" and "Level concrete," "Level concrete" can't begin until "Pour foundation" begins.
- startToFinish — Breakdown Element (B) cannot finish until Work Breakdown Element (A) starts. This dependency type can be used for just-in-time scheduling up to a milestone or the project finish date to minimize the risk of a Work Breakdown Element finishing late if its dependent Work Breakdown Elements slip. If a related Work Breakdown Element needs to finish before the milestone or project finish date, but it doesn't matter exactly when and you don't want a late finish to affect the just-in-time Work Breakdown Element, you can create an SF dependency between the Work Breakdown Element you want scheduled just in time (the predecessor) and its related Work Breakdown Element (the successor). Then, if you update progress on the successor Work Breakdown Element, it won't affect the scheduled dates of the predecessor Work Breakdown Element.

SPEM 2.0 Profile Notation

Modeled as an enumeration.

10 Process Behavior

SPEM 2.0 does not aim to be a generic process modeling language, nor does it even provide its own behavior modeling concepts. SPEM 2.0 rather defines the ability for implementers to choose the generic behavior modeling approach that best fits their needs. It provides specific additional structures to enhance such generic behavior models that are characteristic for describing development processes. The rationale for this approach is to provide the choice and flexibility to SPEM 2.0 specification implementers to select the appropriate behavior model formalism themselves. For example, they could use UML 2 Activity and State Machine diagrams or alternatively BPDM/BPMN.

The Process Behavior package therefore does not introduce its own formalism for behavior models, but instead defines how to generally link process models defined with a third-party representation to the process elements defined in the meta-model package Process Structure. It does this by identifying typical candidate classifiers from such behavior model formalism (in this case classes defined in UML 2's Superstructure) and introduces association for relating SPEM 2.0 classifiers to these elements.

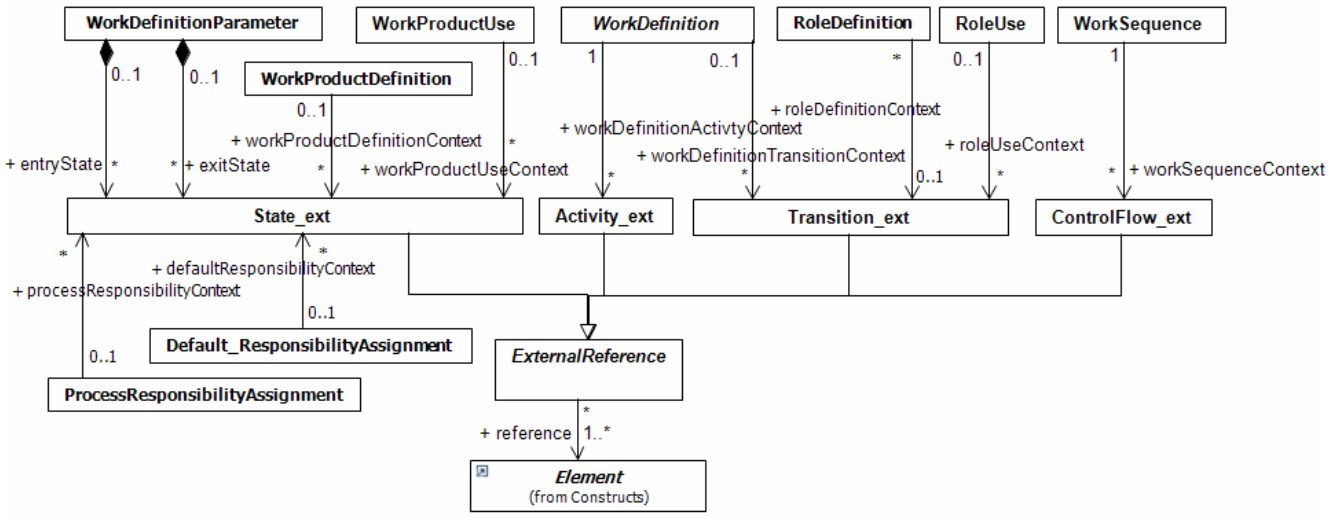


Figure 10.1 - Traceability links from key behavior model abstraction to key Process Elements

Figure 10.1 lists the classes from Process Structure and Method Content that shall be linked to external behavior model classes represented with proxy classes that have been named according to respective UML 2 Superstructure classes using an '_ext prefix. If a link to a different formalism shall be established, the implementer has to map these to respective similar classes. For example, one would link the proxy classes to OMG's BPMN as depicted in Table 10.1. Because BPMN focuses on representing the flow of control, messages, and objects no mapping to State_ext and Transition_ext can be found. However, one could link to more than one formalism to complete the representation of behavior models.

Table 10.1 - Example of mapping SPEM 2.0 External References classes to BPMN

SPEM 2.0 External Reference class	BPMN class
State_ext	n/a
Activity_ext	Activity
Transition_ext	n/a
ControlFlow_ext	SequenceFlow

10.1 Activity_ext

Super Class

External Reference

Description

The Activity_ext represents a reference to a model class in an external behavior model representing a definition of behavior.

10.2 Control Flow_ext

Super Class

External Reference

Description

The ControlFlow_ext represents a reference to a model class in an external behavior model representing a control flow.

10.3 External Reference

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

Description

External Reference is a Classifier that represents an abstract generalization of all classes representing references to external behavior models.

Association Properties

- reference: Element — The reference association pointing to an element representing an element of an external behavior model. If the external behavior model is not represented using the UML Infrastructure library, then the implementer shall define a different way of linking to such elements using the same role name.

10.4 State_ext

Super Class

External Reference

Description

The State_ext represents a reference to a model class in an external behavior model representing a state.

10.5 Transition_ext

Super Class

External Reference

Description

The Transition_ext represents a reference to a model class in an external behavior model representing a transition between states.

10.6 Work Definition Parameter

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

Description

This package extends the Work Definition Parameter class introduced in the Core package with associations to entry and/or exist states for its Work Product Use or Work Definition parameter values.

Association Properties

- entryState: State_ext

  Given that an instance of Work Product Definition or Work Product Use has been specified as an input for a specific Work Definition, then the Entry State attribute specifies the desired state of instances of the referenced work product instance when work on the Work Definition is initiated.
  For some Work Products Definition or Uses, state is expressed in percentage of completion, compliance to work product checklist, informal state descriptions, etc. Others have very specific states expressed as enumerations such as [identified, briefly described, outlined, detailed] for use cases. Other Work Product states relate to some quality measures or lifecycle states such as [reviewed, implemented, tested].

- exitState: State_ex

  Given that an instance of Work Product Definition or Work Product Use has been specified as output for a specific Work Definition, then the Exist State attribute specifies the desired state of instances of the referenced work product instances when work on the Work Definition is finished.

11 Managed Content

The Managed Content package defines the fundamental concepts for managing textual descriptions for process and method content elements. It introduces the abstract class Describable Element that, through package merges, serves as the super class for Process Elements defined in the meta-model package Process Structure (Chapter 9) as well as Method Content Element in the meta-model package Method Content (Section 11.6). Describable Element contains Content Description classes that store the actual textual description.

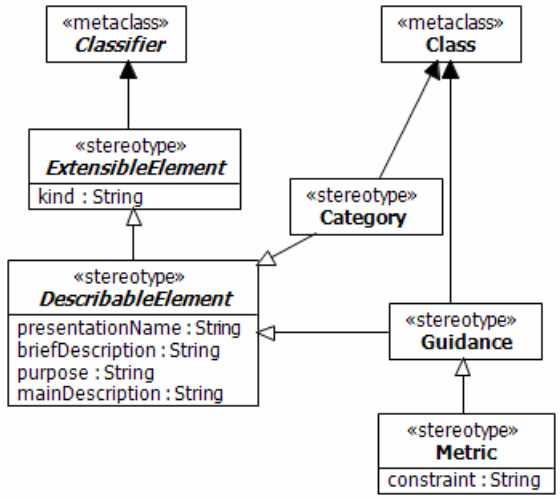


Figure 11.1 - The SPEM 2.0 UML 2 Profile stereotypes defined in the Managed Content package

Figure 11.2 provides an overview to the description parts of Describable Element as well as the classes that have been derived from Describable Element. Note that when Process Structure is not included into the package merge, as in the compliance point “SPEM Method Content,” then Process Element defined here will not be used and will be ignored.

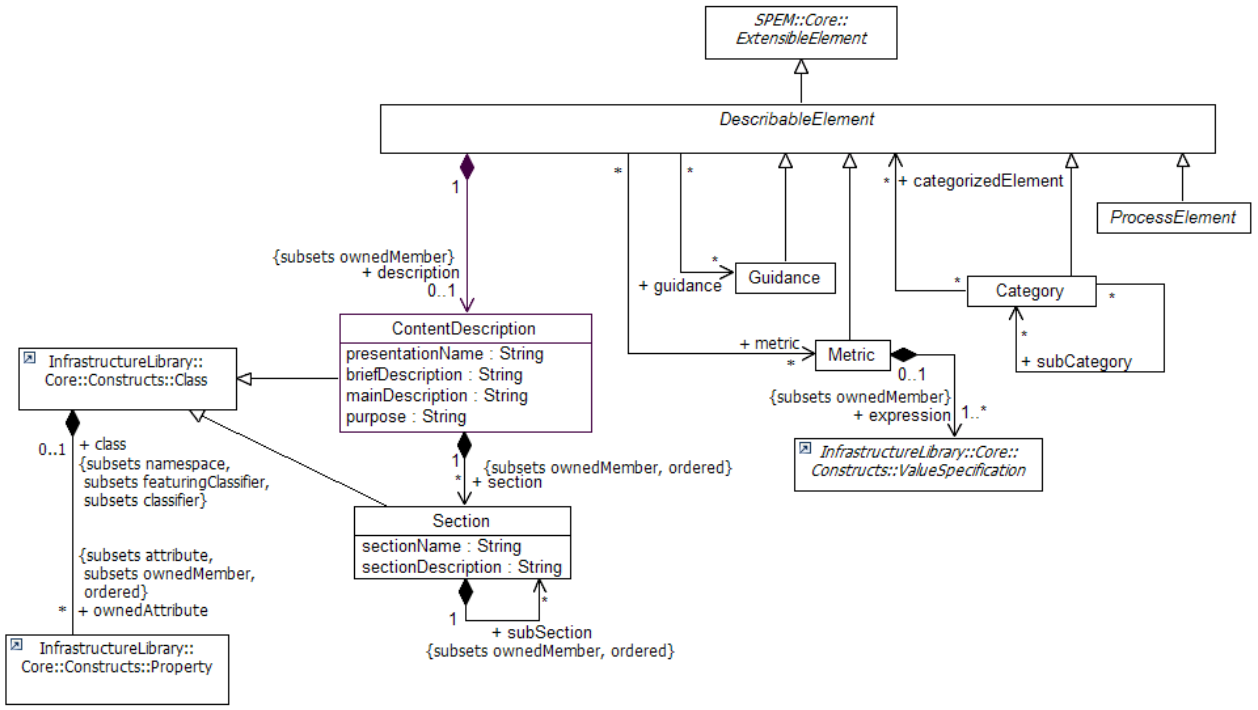


Figure 11.2 - The Describable Element parts and subclasses

11.1 Category

Super Class

Describable Element

Description

A Category is a Describable Element used to categorize, i.e., group any number of Describable Elements of any subtype based on user-defined criteria. Because Categories are Describable Elements themselves, they can be used to recursively categorize other Category instances as well. Categories can also be nested using the subCategory association.

Association Properties

- categorizedElement: ContentElement — A Category groups together any number of Describable Elements (including other Categories).
- subCategory: ContentCategory — A Category can have any number of Categories defined as sub-categories. Therefore, one could define Categories as n-level hierarchies. This relationship does not define a strict nesting, i.e., a Category can be a subcategory of many other Categories.

- kind: CategoryKind — Every Content Category instance shall be classified to be of exactly one Category Kind.

Semantics

Categories can be used to categorize content based on the user's criteria as well as to define whole tree-structures of nested categories allowing the user to systematically navigate and browse method content and processes based on these categories. For example, one could create a Category to logically organize content relevant for the user's development organization departments; e.g., a "Testing" category that groups together all Roles, Work Products, Tasks, and Guidance elements relevant to testing. Another example would be Categories that express licensing levels of the content, grouping freely distributable method content versus content that represents intellectual property and requires a purchased license for use. Whereas Kinds are limited to one Kind instance per meta-model instance and stored as properties of the Extensible Element, Categories store the relationships to Describable Elements. Describable Elements can be categorized by as many Categories as needed. Categories can categorize Categories as well as can be hierarchical.

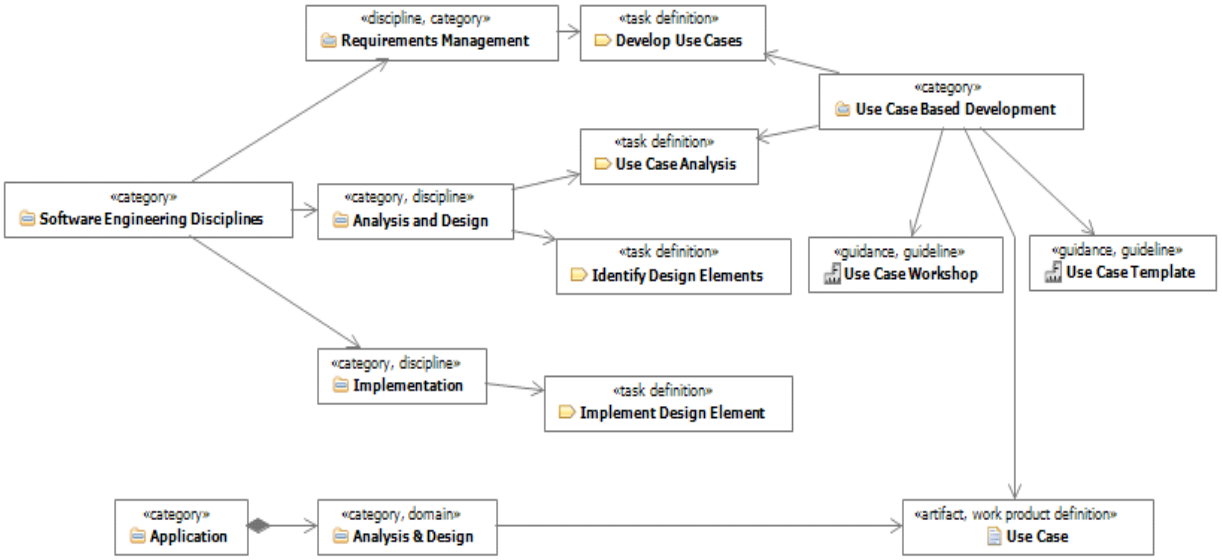


Figure 11.3 - Example for categories and catgorized content

Figure 11.3 shows examples for categorized method content (defined in Chapter 12). It defines the Discipline Grouping "Software Engineering Discipline" that categorizes three Disciplines. The Disciplines categorize Tasks such as "Develop Use Case," "Use Case Analysis," etc. Two of these Tasks have also been categorized by a Category "Use Case Based Development" that a user might have defined to provide a 'filter' on the content related to use cases. The user has also assigned the use case work product from the "Analysis & Design" domain, as well as specific Guidance (template and workshop guideline) to this category.

A CASE tool implementation of SPEM 2.0 would now be able to present a structure such as Figure 11.3 as a navigation tree browser or other navigation structure on the description pages for these Describable Elements, providing hyperlinks along the associations. For example, every Describable Element description would provide a list of the Categories it belongs to, or the other way round Category descriptions would provide lists of all the elements that have been categorized.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
Category	DescribableElement / Class	category	n/a	no	

11.2 Content Description

Super Class

Class (from Constructs in UML 2 Infrastructure)

Description

Content Description is a Class that is used to store the textual description for a Describable Element. It defines standard attributes applicable for all Describable Element subtypes.

Implementers of this specification can subclass Content Description to define their own matching Content Description subtypes that add user-defined description attributes to all instances of a specific Describable Element's Content Description. Additionally, user-defined attributes can be added to individual Content Description instances when the SPEM 2.0 meta-model is instantiated creating UML 2 Infrastructure attributes for the Content Description class accessible via the ownedAttribute inherited from Class. Content Descriptions are typically localized using a resource allocation mechanism for its String type attributes.

Attributes

- presentationName: String — Every Describable Element described by a Content Description has a name (inherited from Named Element in Constructs in UML 2 Infrastructure), which is used for internal references of the element. In addition to name every Describable Element can maintain a presentation name as part of the Content Description, where the presentation name is the externally visible/published name of the element and can be localized.
- briefDescription: String — Every Describable Element shall be briefly described with one or two sentences summarizing the element.
- mainDescription: String — This attribute stores the main descriptive text for the Describable Element. All text that is not part of any of the more specific attributes shall be stored here. If the description is divided into sections using the Section class, then only the text from the 'head' of the content description to the first section will be stored here (similar to a normal document where you can place text between its beginning and its first section heading).
- purpose: String — This attribute summarizes the main reason or rationale for having or performing this Describable Element as part of a Process or Method. It describes what is intended to be achieved with it and why the Process Practitioner should include it.

Association Properties

- section: Section — A Content Description can optionally be structured into Sections. This association is used to decompose the mainDescription attribute into a hierarchy of Sections. Text stored in mainDescription when Sections are defined represents text presented before the first Section.

Semantics

This class provides a container for defining textual descriptions for Describable Elements. It can be extended with user-defined sub-classes adding user defined description attributes.

SPEM 2.0 Profile Notation

Because UML 2 Profile-based models are not suited for large-scale content management application, Content Description and Describable Element have merged into one stereotype: Describable Element.

11.3 Describable Element

Super Class

Extensible Element (from Core)

Description

Describable Element is an Extensible Element that represents an abstract generalization for all elements in SPEM 2.0 that can be documented with textual descriptions. Examples for Describable Elements are Roles or Work Products, which have descriptive text associated that textually define the element as well as providing guidance on how to use it.

Association Properties

- description: ContentDescription — A Describable Element can contain one Content Description element that stores textual descriptions for this Describable Element.
- guidance: Guidance — A Describable Element can be related to many Guidance elements.

Semantics

Describable Element is the superclass for elements in Process Structure as well as Method Content for which concrete textual descriptions are defined in the form of documenting attributes grouped in a matching Content Description instance (see Section 11.2). Describable Elements are intended to be published in method or process publications. Describable Element defines that the element it represents will have content ‘attached’ to it. Content Description is the abstraction for the actual places in which the content is being represented. This separation allows a distinction between core model elements describing the structure of the model from the actual description container providing, for example, the documentation of the Describable Element in different alternatives languages, audiences, licensing levels, etc.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
DescribableElement	ExtensibleElement	n/a	presentatioName, briefDescription, purpose, mainDescription	yes	n/a

11.4 Guidance

Super Class

Describable Element

Description

Guidance is a Describable Element that provides additional information related to Describable Elements. The particular Guidance should be classified with Kinds (Section 8.2) that indicates a specific type of guidance for which perhaps a specific structure and type of content is assumed. Examples for Kinds for Guidance are Guidelines, Templates, Checklists, Tool Mentors, Estimates, Supporting Materials, Reports, Concepts, etc. See Section 18.3 for more examples and definitions for typical Kinds for Guidance.

Semantics

No additional semantics.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
Guidance	DescribableElement / Class	guidance	n/a	no	SPEM 1.1 backwards compatibility icon:

11.5 Metric

Super Class

Describable Element

Description

A Metric is a special Describable Element that contains one or more constraints that provide measurements for any Describable Element. Because Metric is an Extensible Element, different Kinds (Section 8.2) can be defined for Metrics to distinguish different groups of Metrics such as Productivity, Quality, or Scale.

Association Properties

- expression: ValueSpecification — A Metric defines one or more Value Specification (from UML 2 Infrastructure Constructs). The property subsets the inherited ownedMember property from UML 2 Infrastructure's Namespace in Constructs.

Semantics

A Metric defines a standard measurement for instances of a Describable Element in SPEM 2.0. For example, a process engineer can define Metrics for Work Definitions such as Activities (estimated effort in man hours), Metrics for Work Products (quality averages such as error per klocs), or Metrics for Roles (costs per hour; cost per delivered results). A Metric is documented with Content Descriptions associated to the Metric as well as formalized using instances of the UML 2 Value Specification class. Metrics can be qualified with Kinds.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
Metric	Guidance	metric	constraint	no	

11.6 Section

Super Class

Class (from Constructs in UML 2 Infrastructure)

Description

A Section is a special Class that represents a structural subsection of a Content Description's mainDescription attribute. It is used for large scale documentation of Describable Elements organized into sections, as well as to flexibly add new Sections to Describable Elements using contribution variability (added to the Section concept for Method Plug-ins in Section 14.9).

Attributes

- sectionName: String — This attributes stores the name or the header of the section.
- sectionDescription: String — This attribute stores the description text for a Content Description's Section.

Association Properties

- subSection: Section — Sections can be further decomposed into n levels of sub-sections.

Semantics

This class represents Content Descriptions structured into Sections.

SPEM 2.0 Profile Notation

Not available.

12 Method Content

The Method Content package defines the core elements of every method such as Roles, Tasks, and Work product Definitions.

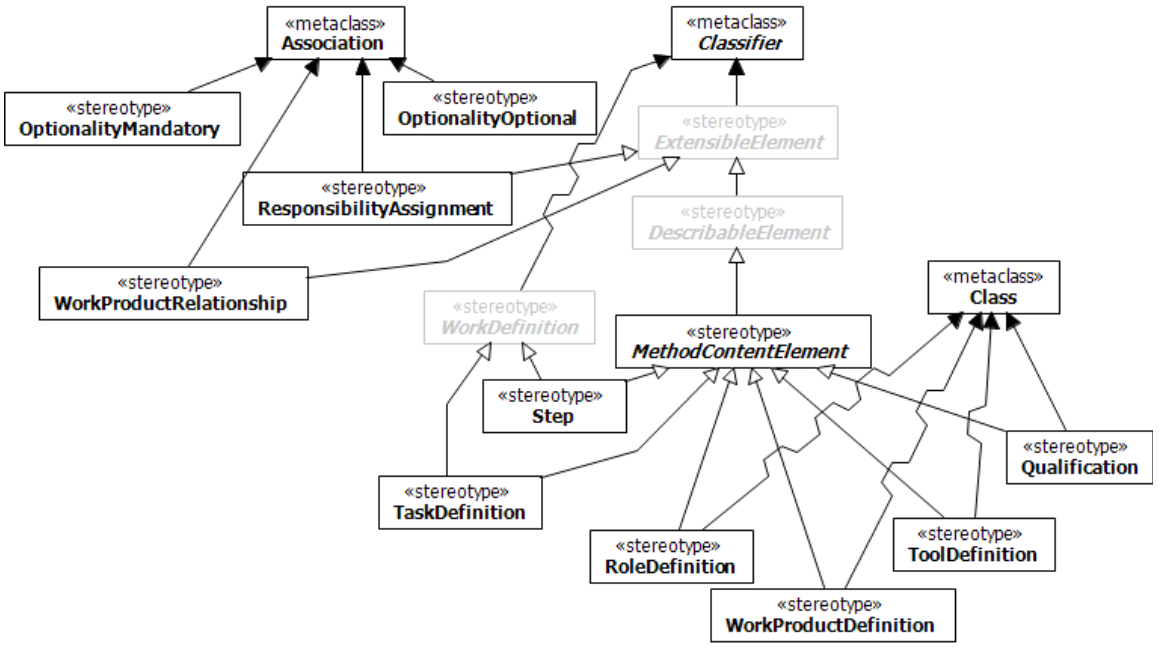


Figure 12.1 - The SPEM 2.0 UML 2 Profile stereotypes defined in the Method Content package

Figure 12.2 shows the taxonomy of the elements defined in this package. All Method Content, with the exception of Step, is derived from the class Describable Element. Step is not a Describable Element because it is defined as a special Section classifier (defined in Section 11.6 as a part of Content Description). Hence, Step already is a description for content and therefore does not require being a Describable Element with its own Content Description part. Step and Task Definition are derived from an abstraction called Work Definition (introduced in the meta-model package Core, Chapter 8), which represents behavioral descriptions for units of work.

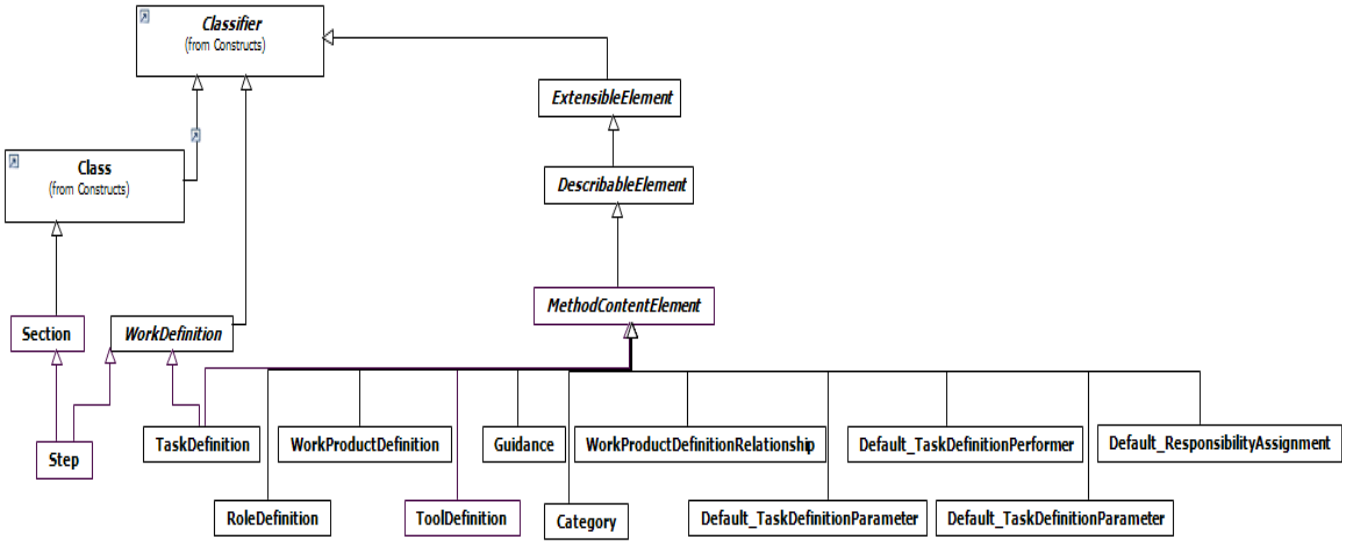


Figure 12.2 - Taxonomy of Core Describable Elements

Method Content is fundamentally described by defining Task Definitions organized into Steps, having Work Product Definitions as input and output, and performed by Roles Definitions. Role Definitions define important responsibility relationships to work products.

Figure 12.3 depicts these core elements with their relationships model as subclasses of Method Content Element. The relationships are labeled with a “Default_” prefix, because they describe the ‘standard’ or ‘normal’ way of performing the method described; i.e., doing the work under ‘normal’ circumstances, with ‘normal’ being what the method author perceives as the most general case. However, different processes would apply these elements in different parts of a lifecycle differently (cf. with Section 6.3.1 and Chapter 13) and then might define changes on these relationships. For example, if a Task is performed in more than one iteration of an iterative lifecycle model, different Roles could be added to (or removed from) performing the Task as well as different inputs might be considered at different points in time. Such changes would be defined on process specific Method Content Uses that reference and have been defined based on these method Describable Elements (see Chapter 13 for more details).

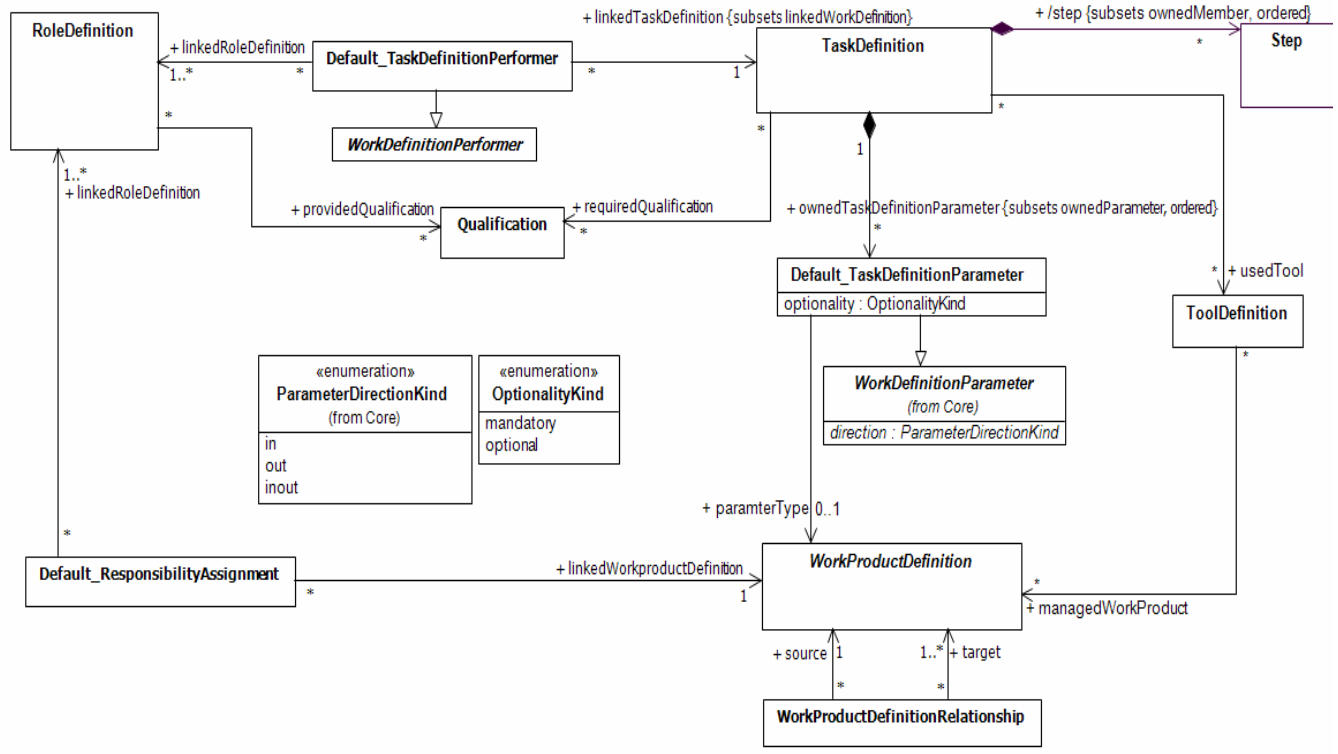


Figure 12.3 - Key Method Content Elements and their relationships

12.1 Default Responsibility Assignment

Super Class

Method Content Element

Description

A Default Responsibility Assignment is a Method Content Element that represents a relationship between instances of Role Definition and Work Product Definition. An instance of the Default Responsibility Assignment links one or more Role Definition instances to exactly one Work Product Definition.

Association Properties

- linkedRoleDefinition: RoleDefinition — A Process Responsibility Assignment links to one or more Role Definition.
- linkedWorkProductDefinition: WorkProduct-Definition — A Process Responsibility Assignment links to exactly one Work Product Definition.

Semantics

The Process Responsibility Assignment links Role Definitions to Work Product Definitions, indicating that the Role Definition has a responsibility relationship with the Work Product Definition. The Kind of responsibility of the Role Definition for the Work Product Definition needs to be defined by Kind (Section 8.2) class instances that qualify the Default Responsibility Assignment. The popular RACI-VS diagram defines a set of commonly used Kinds that cannot only be applied for the Default Task Definition Performer, but also often used for work product responsibility: Responsible, Accountable, Consulted, Informed, Verifies, and Signs.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
ResponsibilityAssignment	ExtensibleElement / Association	responsible	n/a	no	n/a

12.2 Default Task Definition Parameter

Super Class

Work Definition Parameter

Description

A Task Definition Parameter is a special Work Definition Parameter that uses Work Product Definitions as well as adds an Optionalilty attribute.

Attributes

- Optionality: OptionalityKind — This attribute represents whether the input or output parameter is optional, i.e., mandatory to provide to the Task or to be produced by the Task respectively.

Association Properties

- parameterType : WorkProductDefinition — This association links zero or one Work Product Definitions instances to a parameter. Task Definitions can leave the type specification open and not specify a concrete Work Product Definition at all.

SPEM 2.0 Profile Notation

Use Work Definition Parameter (see Section 8.5).

12.3 Default Task Definition Performer

Super Class

Work Definition Performer (from Core)

Description

A Default Task Definition Performer is a Work Definition Performer that represents a relationship between Task Definition instances and Role Definition instances. An instance of Default Task Definition Performer links one or more Role Definition instances to one Task Definition instance.

Association Properties

- linkedTaskDefinition: TaskDefinition — A Default Task Definition Performer links to exactly one Task Definition. The linked Task Definition property subsets the linkedWorkDefinition property from the Work Definition Performer defined in Core.
- linkedRoleUse: RoleUse — A Default Task Definition Performer links to one or more Role Definition.

Semantics

The Default Task Definition Performer links Role Definition to Task Definitions indicating that these Role Definition instances participate in the work defined by the Task Definition in one way or another. The kind of involvement of the Role Definition in the Task Definition needs to be defined by Kind (Section 8.2) class instances that qualify the Default Task Definition Performer instances. Typical examples for Kinds of Default Task Definition Performer would be Primary Performer, Additional Performer, Assisting Performer, Supervising Performer, Consulted Performer, etc. The popular RACI-VS diagram defines another set of commonly used Kinds for the Default Task Definition Performer: Responsible, Accountable, Consulted, Informed, Verifies, and Signs.

SPEM 2.0 Profile Notation

Use Work Definition Performer (see Section 8.6).

12.4 Method Content Element

Super Class

Describable Element

Description

Method Content Element is an abstract Describable Element that represents an abstract generalization for all Method Content Elements in SPEM 2.0. Because Method Content Element derives from Describable Element, it contains textual descriptions.

Semantics

No additional semantics define. See subclasses.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
MethodContentElement	DescribableElement	n/a	n/a	yes	n/a

12.5 Optionality Kind

Super Class

n/a: Enumeration

Description

This enumeration provides the values for the Task Definition Parameter attribute optionality.

Enumeration Literals

- mandatory — It is mandatory to provide the Work Product Definition specified in this parameter as input or to provide an instance of the Work Product Definition as output respectively.
- optional — It is optional to provide the Work Product Definition specified in this parameter as input or to provide an instance of the Work Product Definition as output respectively.

SPEM 2.0 Profile Notation

Stereotype	Metaclass	Keyword	Properties	Abstract	Icon
OptionalityMandatory	Association	mandatory	n/a	no	n/a
OptionalityOptional	Association	optional	n/a	no	n/a

12.6 Qualification

Super Class

Method Content Element

Description

Qualification is a Method Content Element that documents zero or more required qualifications, skills, or competencies for Role and/or Task Definitions. In addition to informally describing the qualification using its Content Element documentation properties, Qualification can be further categorized by defining specific Kinds.

Semantics

A Qualification documents one specific skill or competency that is used to model and represent the qualifications provided by instances of a Role Definition and/or the qualifications required for the performance of a Task. These qualifications can be used to find and map roles for tasks when assembling method content and assigning organization specific roles to these tasks dynamically. Qualifications can also be used to find individuals (i.e., people) as instances of the Role Definition instances. For example, Qualifications of the Kind “Soft Skills” associated to the “System Analyst” Role Definition could be “captures stakeholder needs,” “selects elicitation techniques,” “adapts elicitation techniques,” “negotiates scope,” etc.

A Role Use (see Section 13.13) can select a sub-set of valid Qualifications for the use of the Role Definitions in the context of a particular Activity. One can then express that for the Role Use within a particular Activity, only the “selects elicitation techniques” Qualification of the “System Analyst” Role Definition is required.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
Qualification	MethodContentElement / Class	qualification	n/a	no	n/a

12.7 Role Definition

Super Class

Method Content Element

Description

A Role Definition is a Method Content Element that defines a set of related skills, competencies, and responsibilities. Roles are used by Task Definitions to define who performs them as well as to define a set of Work Product Definitions they are responsible for.

Association Properties

- providedQualification: Qualification — Provides a list of qualifications that the role typically provides. This list can be mapped against the required qualifications list defined for Task Definitions (see Section 12.9). The qualifications need to be present by individual that are represented as instances of instances of the Role Definitions.

Semantics

A Role Definition defines a set of related skills, competencies, and responsibilities of an individual or a set of individuals. Roles are not individuals or resources. Individual members of the development organization will wear different hats, or perform different roles. The mapping from individual to Role, performed by the project manager when planning and staffing for a project, allows different individuals to act as several different roles, and for a role to be played by several individuals (also refer to Composite Role in Section 13.3).

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
RoleDefinition	MethodContentElement / Class	role definition	n/a	no	SPEM 1.1 backwards compatibility icon:

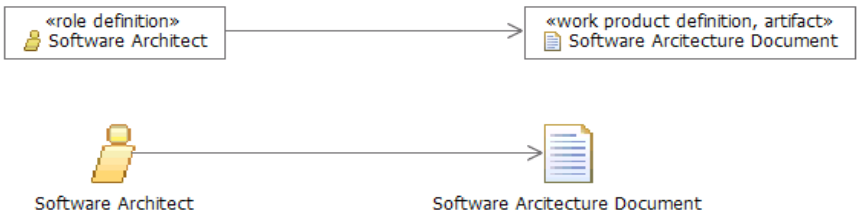


Figure 12.4 - Example for a UML2 diagram with role stereotypes and artifact responsibility

12.8 Step

Super Class

Section

Work Definition

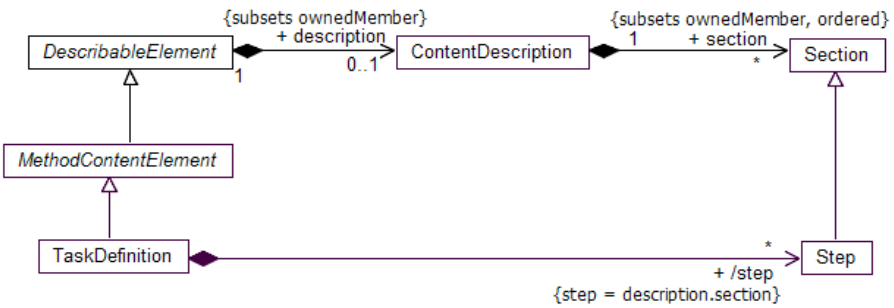


Figure 12.5 - Steps represent Sections of Tasks

Description

A Step is a Section and Work Definition that is used to organize a Task Definition's Content Description into parts or subunits of work. Steps inherit the subSection decomposition from Section and can therefore describe sub-Steps nested into Steps.

Semantics

A Step describes a meaningful and consistent part of the overall work described for a Task Definition. The collection of Steps defined for a Task Definition represents all the work that should be done to achieve the overall development goal of the Task Definition. Not all steps are necessarily performed each time a Task is invoked in a Process (see Task Use defined in Section 13.14), so they can also be expressed in the form of alternate 'flows' of work. Different ways of achieving the same development goal can then be 'assembled' by selecting different combinations of steps when applying the Task Definition in a Process. Typical kinds of steps that a Task Definition author should consider are:

- Thinking steps: where the individual roles understand the nature of the task, gather and examine the input artifacts, and formulate the outcome.
- Performing steps: where the individual roles create or update some artifacts.
- Reviewing steps: where the individual roles inspect the results against some criteria.

SPEM 2.0 Profile Notation

As there is no Content Description and Section stereotype, Step has been directly derived from Method Content Element.

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
Step	MethodContentElement, WorkDefinition	step	n/a	no	

12.9 Task Definition

Super Class

Method Content Element

Work Definition

Description

A Task Definition is a Method Content Element and a Work Definition that defines work being performed by Roles Definition instances. A Task is associated to input and output Work Products. Inputs are differentiated in mandatory versus optional inputs. The relationships to Work Products via Work Definition Parameters are not instantiatable/variable-like parameters. They rather express (hyper-)links to the descriptions of the work products types that are related to the Task as inputs and outputs. In other words, these associations are not intended to be used to capture which concrete instances will be passed along when instantiating the method in a project. All of the Task Definition's default associations and Parameters can be overridden in an actual process definition (see Chapter 13).

Association Properties

• ownedTaskDefinitionParameter: Default_TaskDefinitionParameter	This ordered association subsets ownedParameter from Work Definition. Tasks Definitions manage Task Definition Parameter instances that are subclasses of the Work Definition Parameters. Task Definition Parameter adds an optionality property to the parameter.
• usedTool: ToolDefinition	A Task Definition can recommend a specific set of tools to be used to support the Task.
• /step: Step	This derived association is a shortcut for navigation along the association properties self.description.section.
• requiredQualification: Qualification	Provides a list of qualifications that the task typically requires to be performed by one or more roles. This list can be mapped against the provided qualifications list defined for Role Definitions (see Section 12.7).

Semantics

A Task Definition describes an assignable unit of work. Every Task Definition is assigned to specific Role Definitions. The granularity of a Task Definition is generally a few hours to a few days. It usually affects one or only a small number of work products. A Task Definition is used as an element of defining a process. Task Definitions are further used for planning and tracking progress; therefore, if they are defined too fine-grained, they will be neglected, and if they are too large, progress would have to be expressed in terms of a Task Definition's parts (e.g., Steps, which is not recommended).

A Task Definition has a clear purpose in which the performing roles achieve a well defined goal. It provides complete step-by-step explanations of doing all the work that needs to be done to achieve this goal. This description is completely independent of when in a process lifecycle the work would actually be done. It therefore does not describe when you do what work, but describes all the work that gets done throughout the development lifecycle that contributes to the achievement of this goal. When the Task Definition instance is applied in a process, then this process application (defined as Task Use in Section 13.14) provides the information of which pieces of the Task Definition will actually be performed at any particular point in time. This assumes that the Task Definition will be performed in the process over and over again, but each time with a slightly different emphasis on different steps or aspects of the task description (also see Section 6.3.1 summarizing the difference between Method Content and Process).

For example, a Task Definition such as "Develop Use Case Model" describes all the work that needs to be done to develop a complete use case model. This would be comprised of the identification and naming of use cases and actors, the writing of a brief description, the modeling of use cases and their relationships in diagrams, the detailed description of a basic flow, the detailed description of alternatives flows, performing of walkthroughs workshops and reviews, etc. All of these parts contribute to the development goal of developing the use case model, but the parts will be performed at different points in time in a process. Identification, naming, and brief descriptions would be performed early in a typical development process versus the writing of detailed alternative flows that would be performed much later. All these parts or steps within the same Task define the "method" of Developing a Use Case Model. Applying such a method in a lifecycle (i.e., in a process) is defining which steps are done when going from one iteration to the next.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
TaskDefinition	MethodContentElement, WorkDefinition	task definition	n/a	no	SPEM 1.1 backwards compatibility icon:

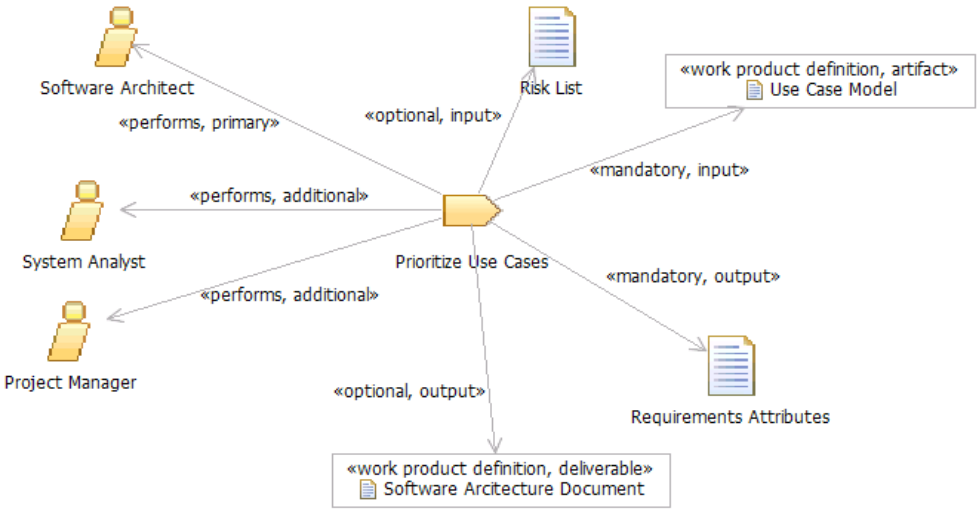


Figure 12.6 - Example Task with related content elements represented using the UML 2 SPEM 2.0 Profile

12.10 Tool Definition

Super Class

Method Content Element

Description

A Tool Definition is a special Method Content Element that can be used to specify a tool's participation in a Task Definition.

Association Properties

• managedWorkProduct: WorkProductDefinition	A Tool can manage instances of one or more Work Product Definition. For example, a Tool can be modeled that specializes in managing Use Case Models or another Tool that manages Analysis and Design Models.

Semantics

A Tool Definition describes the capabilities of a CASE tool, general purpose tool, or any other automation unit that supports the associated instances of Role Definitions in performing the work defined by a Task Definition. A Tool Definition can identify a resource as useful, recommended, or necessary for a task's completion. The Task Definition can refer to the associated list of Tool Definitions to clarify their role.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
ToolDefinition	MethodContentElement / Class	tool definition	n/a	no	

12.11 Work Product Definition

Super Class

Method Content Element

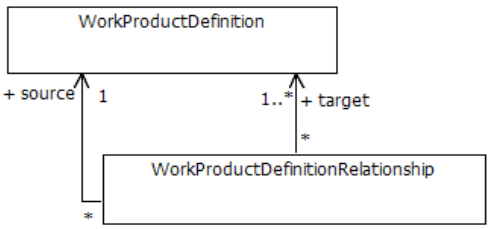


Figure 12.7 - Work Product taxonomy

Description

Work Product Definition is Method Content Element that is used, modified, and produced by Task Definitions. Work Product Definitions can be related to other Work Product Definitions via the Work Product Definition Relationship.

Semantics

Work Products are in most cases tangible work products consumed, produced, or modified by Tasks. They may serve as a basis for defining reusable assets. Roles use Work Products to perform Tasks and produce Work Products in the course of performing Tasks. Work Products are the responsibility of Role Definitions, making responsibility easy to identify and

understand, and promoting the idea that every piece of information produced in the method requires the appropriate set of skills. Even though one Role Definition might “own” a specific type of Work Product, other roles can still use the Work Product for their work, and perhaps even update them if the Role Definition instance has been given permission to do so.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
WorkProductDefinition	MethodContentElement / Class	work product definition	n/a	no	SPEM 1.1 backwards compatibility icon:

12.12 Work Product Definition Relationship

Super Class

Method Content Element

Description

A Work Product Definition Relationship expresses a general relationship among Work Products Definitions. Kind (Section 8.2) class instances shall be used to specify the nature of this relationship.

Association Properties

- source: Work Product Definition — This association links to the exact one source of the Work Product Definition Relationship.
- target: Work Product Definition — This association links to one or more targets of the Work Product Definition Relationship.

Semantics

Work Product Definition Relationship has almost the same semantics as Work Product Use Relationship (see Section 9.12 for details). However, it is used to represent generic or default relationships among Work Product Definitions that have been defined in the general methods around these Work Products. They are generic and completely lifecycle independent. Work Product Use Relationships are more accurate towards a concrete application of Work Product Definitions in a specific process or even specific activities (such as phases or iterations within a Process).

SPEM 2.0 Profile Notation

SPEM 2.0 users need to define their own stereotypes for the Kind classes that they produce. The Base Plugin defines a set of typical Kinds and stereotypes in Section 18.5.

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
WorkProductRelationship	ExtensibleElement / Association	work product relationship	n/a	no	n/a

13 Process with Methods

SPEM 2.0 separates reusable core method content from its application in processes. A Development Process defines the structured work definitions that need to be performed to develop a system, e.g., by performing a project that follows the process. Such structured work definitions delineate the work to be performed along a timeline or lifecycle and organize it in so-called breakdown structures.

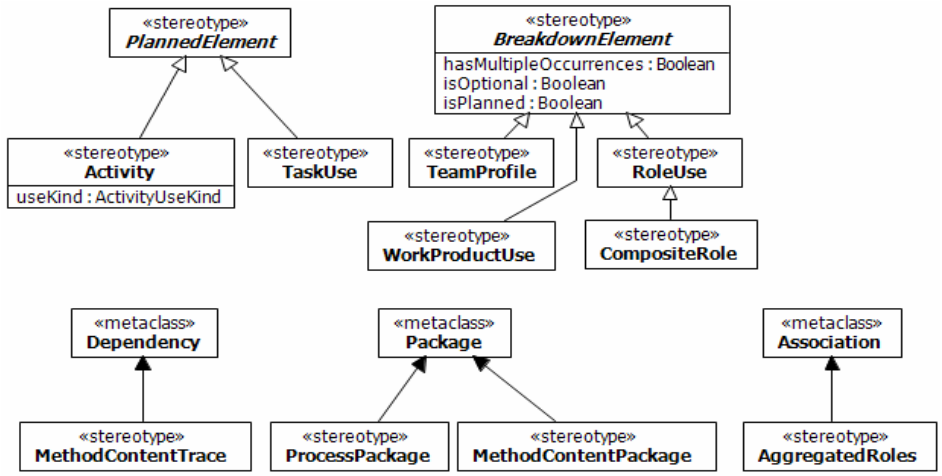


Figure 13.1 - The SPEM 2.0 UML 2 Profile stereotypes defined in the Process with Methods package

Task Definitions (introduced as part of the core method content in Section 12.9) provide complete step-by-step explanations for work that needs to be performed to achieve a particular development goal. Tasks do not look at the 'big picture' of how the work described relates to other Tasks' work. They have been defined as reusable core method content independent of their placement within a development lifecycle. All that is defined for a Task are the types of the input and output work products, the concrete steps to perform, as well as what skills are required to perform the work (by defining and relating to Roles). Which Tasks have to precede and succeed a Task as well as what steps to focus on at what point of time in the lifecycle are intentionally not defined for core method content. The goal is to provide Tasks that can be reused for many different development situations.

The scope of a process is to provide extended as well as concrete breakdown structures for a specific development situation. Therefore, a process with methods takes reusable core method content elements such as Tasks and Work Product Definitions and relates them into partially-ordered sequences that are customized to specific types of projects. To do so, SPEM 2.0's Process with Methods package provides data structures that reflect the industry's best practices such as the separation of Actions from Activities found in UML 2 Superstructure. This section refines the Breakdown Structures introduced in the Process Structure package (Chapter 9) with mappings to Method Content Elements. It also refines the notion of Activity to represent higher level groupings of work that now also comprise references to Task Definitions (which are similar to UML 2 Activities), called Task Uses (which are similar to UML 2 Actions).

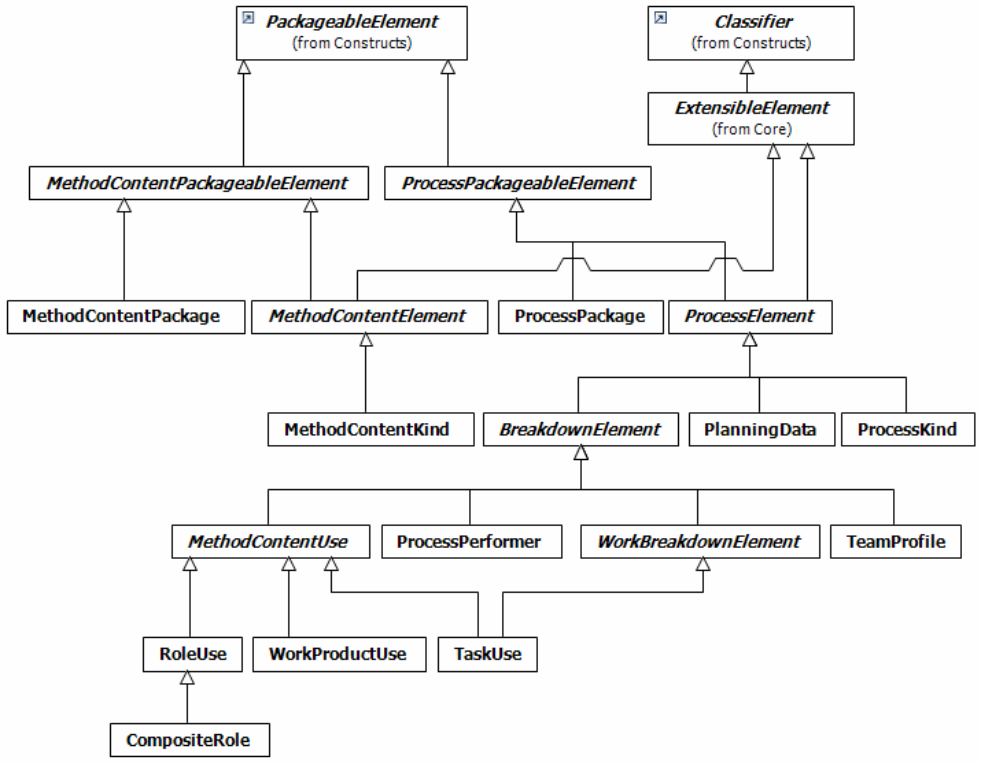


Figure 13.2 - Taxonomy of the meta-model classes defined in the Process with Methods meta-model package

Figure 13.2 depicts the taxonomy of classifiers introduced or refined in the Process with Methods package. It shows that a few new abstract classes have been introduced that serve as new superclasses to classes introduced in other meta-model packages such a Process Structure and Method Content. These have been added to enforce the separation of method content from processes. For example, the classes Method Content Packageable Element and Process Packageable Element are superclasses for all classes introduced so far, categorizing each class into either method content or process. Two redefinitions of the UML 2 Infrastructure class Package, which served as the packaging concept in the other meta-model packages, have been created here to ensure that instances of the different categories are not packaged together.

The key static structure for describing processes is the breakdown structure defined around the class Breakdown Element, which detailed taxonomy and structural relationships are depicted in Figure 13.3. This diagram presents the different types of Breakdown Elements that have been extended in this meta-model package as well as their key new relationships introduced here.

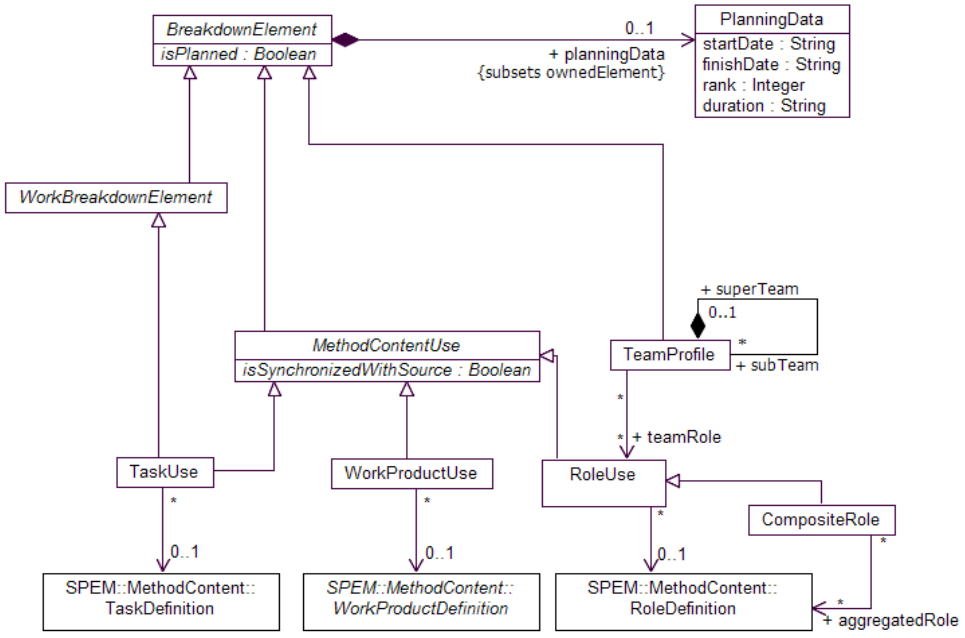


Figure 13.3 - Taxonomy and key relationships of Breakdown

13.1 Activity

Super Class

Work Breakdown Element

Work Definition

Semantics

Activity from Process Structure (Section 9.1) is extended in this meta-model package with the ability to also contain Task Uses, Team Profiles, and Composite Roles.

Activity represents a grouping of nested Breakdown Elements such as other Activity instances, Task Uses, Role Uses, Milestones, etc. It is not just a ‘high-level’ grouping of work such as Work Definitions as in other similar meta-models. It also aims to be a grouping for all different kinds of Breakdown Elements defining a namespace for these elements. The goal for this approach is that instances of specific Breakdown Element instances need to define different relationships and textual documentation properties for occurrences in different activities. For example, a SPEM 2.0 user could create three instances of Role Use that represent a Role Definition called “System Analyst” for three different activities. In each of these activities, the different Role Use instances all representing the same Role Definition “System Analyst” could be modeled with different relationships, such as different responsibilities for Work Products, to represent the fact that the System Analyst has to focus on different responsibilities in different activities (e.g., he might be responsible for quite different work product in an early phase of a project than in a later phase; with phases modeled as Activities).

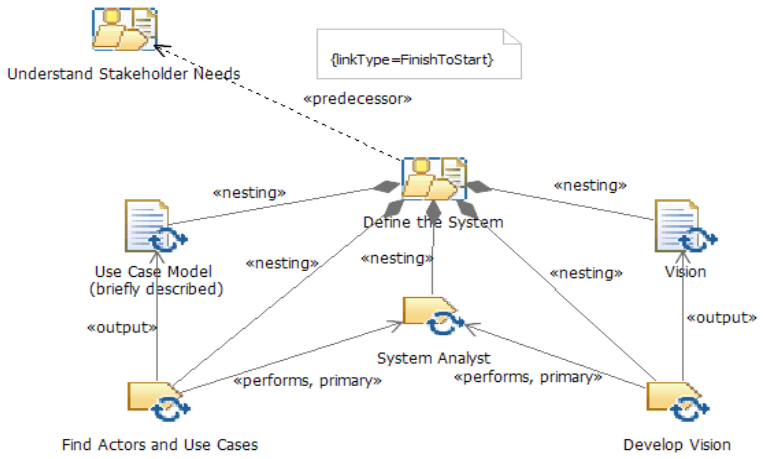


Figure 13.4 - Example for an Activity 'Define the System' with associations

Figure 13.3 depicts a SPEM 2.0 Profile presentation of an Activity called "Define the System" that groups two Task Uses, two Work Product Uses, and one Role Use. It also defines that Activity "Understand Stakeholder Needs" is its predecessor that has to be completed before work on "Define the System" can start (Tagged Value: "linkType=FinishToStart," cf. with Work Sequence in Section 9.11). The diagram is an alternative UML-based presentation. A more common presentation of Activities and work breakdown structures is depicted in Figure 13.4, which shows the same Activity "Define the System."

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
Activity	WorkDefinition, Planned Element / Action	activity	n/a	no	SPEM 1.1 backwards compatibility icon:

13.2 Breakdown Element

Super Class

Process Element

Description

Breakdown Element is an abstract generalization for any type of Process Element that is part of a breakdown structure. It defines a set of properties available to all of its specializations.

Attributes

- isPlanned: Boolean = true

A key application for Development Processes expressed with Breakdown structures is to generate a project plan from it. A process as defined in SPEM 2.0 (cf. Section 9.1) is a multi-dimensional structure defining what work is being performed at what time by which roles using which input and producing what outputs. A project plan as it is represented in project planning tools such as IBM Rational Portfolio Manager or Microsoft Project normally does not need all this information and is normally limited to just representing a subset. For example, a typical MS Project plan only represents the work breakdown consisting of tasks and activities (sometimes referred to as summary tasks). It does not show the input and output Work Products for a task, but it can show which roles shall be staffed for performing the task. However, such role allocation needs to be replaced with concrete resources when instantiating the plan for a concrete project. Sometimes project plans can then again be organized differently by organizing work by deliverables in which Work Products are mapped to the plan's summary tasks and Task that have these work products as output mapped below such as summary task. Therefore, a process can make recommendations about which elements to include and which to exclude when generating a plan.

When the isPlanned attribute is set to False for an instance of a Breakdown Element, then this element shall not be included when a concrete project plan is being generated from the breakdown structure that contains this element.

Association Properties

- planningData: PlanningData

The PlanningData class factors out specific optional data needed for representing planning templates. This association allows accessing planning data if it is stored for the Breakdown Element.

Semantics

No additional semantics. See subclasses.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
BreakdownElement	ProcessElement	n/a	hasMultipleOccurences, isOptional, isPlanned	yes	n/a

13.3 Composite Role

Super Class

Role Use

Description

A Composite Role is a special Role Use that relates to more than one Role Definition. It represents an aggregation of Roles Definition references for an Activity with the main purpose of simplification, i.e., reducing the number of roles defined in method content for a process.

Association Properties

- aggregatedRole: Role — This association lists all the Roles Definitions represented by the Composite Role.

Semantics

A Composite Role is a grouping of Role Definitions that can be used in an Activity or Process to reduce the number of Roles defined in Method Content. A typical application would be a process for a small team in which a standard set of Roles Definitions from the method content would be all performed by one or more Role Use instance. By using Composite Roles, the process would suggest a typical clustering of Role Definitions to just one instantiatable element in the process. A Composite Role instance would then perform all Tasks defined for the Roles Definitions it refers to.

Figure 13.15 shows an example for Composite Role named Developer that groups together the Implementer and Tester Roles. Every time one of the Role Implementer or Tester would be used within the breakdown structure of Figure 13.15, Developer is used instead. Hence, if a Task Use would be added to the Process represented in Figure 13.15 that has Implementer or Tester defined to be the primary performer as a Role Use, this Role Use would be substituted automatically by a Composite Role instance Developer that links back to either Tester or Implementer (or both if both were listed as Task performers).

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
CompositeRole	RoleUse	composite role	n/a	no	
AggregatedRole	Association	aggregated role	n/a	no	n/a

13.4 Method Content Kind

Super Class

Kind (from Core)

Method Content Element

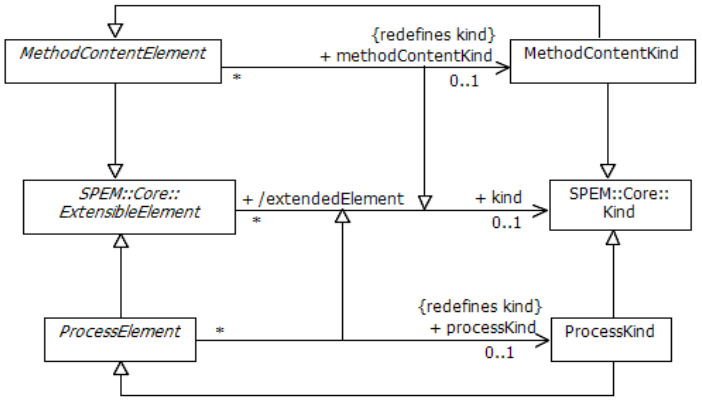


Figure 13.5 - Refined Method Content and Process Kinds

Description

Method Content Kind is a Kind and Method Content Element that represents a method content specific refinement of the Kind class defined in Core (Section 8.2). Only Method Content Elements can be used for Method Content Kinds. Method Content Elements can be packaged in Method Content Packages (Section 13.5).

Semantics

See Kind (Section 8.2).

13.5 Method Content Package

Super Class

Package (from Constructs in UML 2 Infrastructure)

Method Content Packageable Element

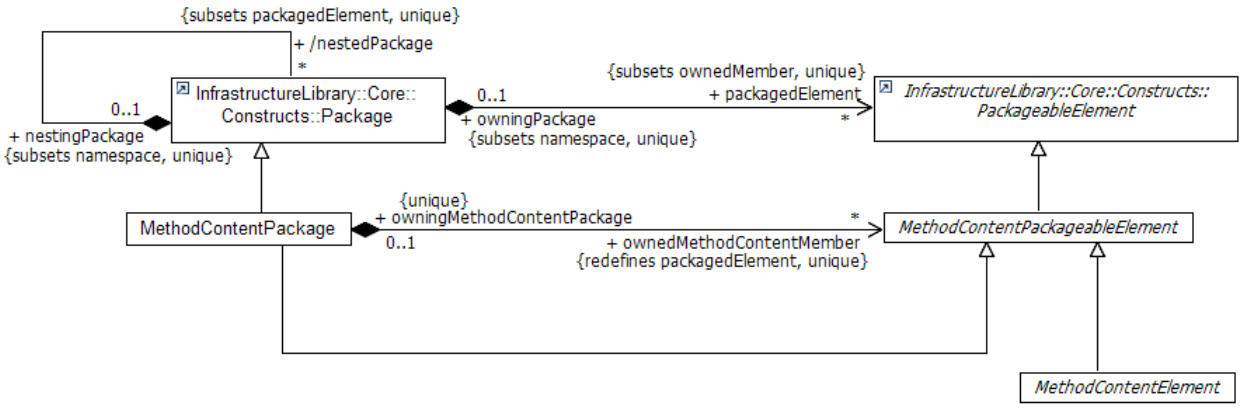


Figure 13.6 - ContentPackage redefines its inherited owned Members association to only allow Method Content Elements

Description

A Method Content Package is a Method Content Packageable Element and Package that contains Method Content Elements only. Examples for Method Content Element are Work Product Definition, Task Definition, Role Definition, etc.

Association Properties

• ownedMethodContentMember: Method-ContentPackageableElement	A Method Content Package contains Method Content Packageable Elements. A Method Content Element instance can be part of only one package instance.

Semantics

A key separation of concerns in SPEM 2.0 is the distinction between Method Content and Process as outlined in Section 6.3.1. This separation is enforced by special package types, which do not allow the mixing of method content with processes. See Section 13.9 for the definition of Process Package.

Method Content Elements are organized in Method Content Packages to structure large scales of method content as well as to define a mechanism for reuse. Method Content Elements from one package can reuse element from other packages by defining an imports dependency (defined in the UML 2 Infrastructure) between them. For example, a Work Product Definition defined in one package can be used as an input for Task Definitions defined in other packages. Reusing it from one common place (i.e., the package in which it has been defined) ensures that no redundant definitions of the same elements are required. Also, maintenance of method content is greatly improved as changes can be performed in only one place.

SPEM 2.0 Profile Notation

Stereotype	Metaclass	Keyword	Properties	Abstract	Icon
MethodContentPackage	Package	method content package	n/a	no	

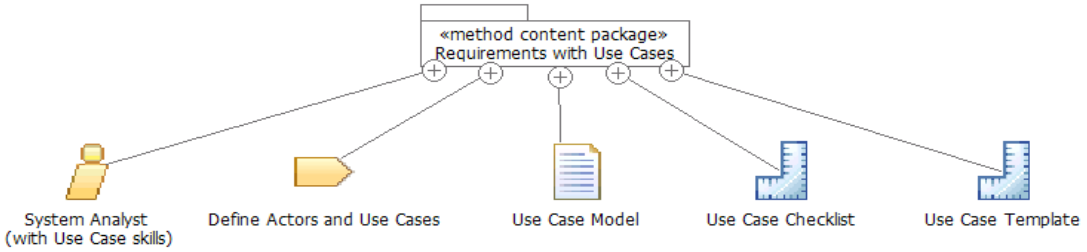


Figure 13.7 - Example for a Content Package with its Content Elements

13.6 Method Content Packageable Element

Super Class

Packageable Element (from Constructs in UML 2 Infrastructure)

Description

Method Content Packageable Element is an abstract generalization for Method Content Package and Method Content Element supporting the redefinition of the packagedElement association of Method Content Package inherited from the UML 2 class.

Semantics

Method Content Packageable Element represents an element that can be packaged in a Method Content Package.

13.7 Method Content Use

Super Class

Breakdown Element

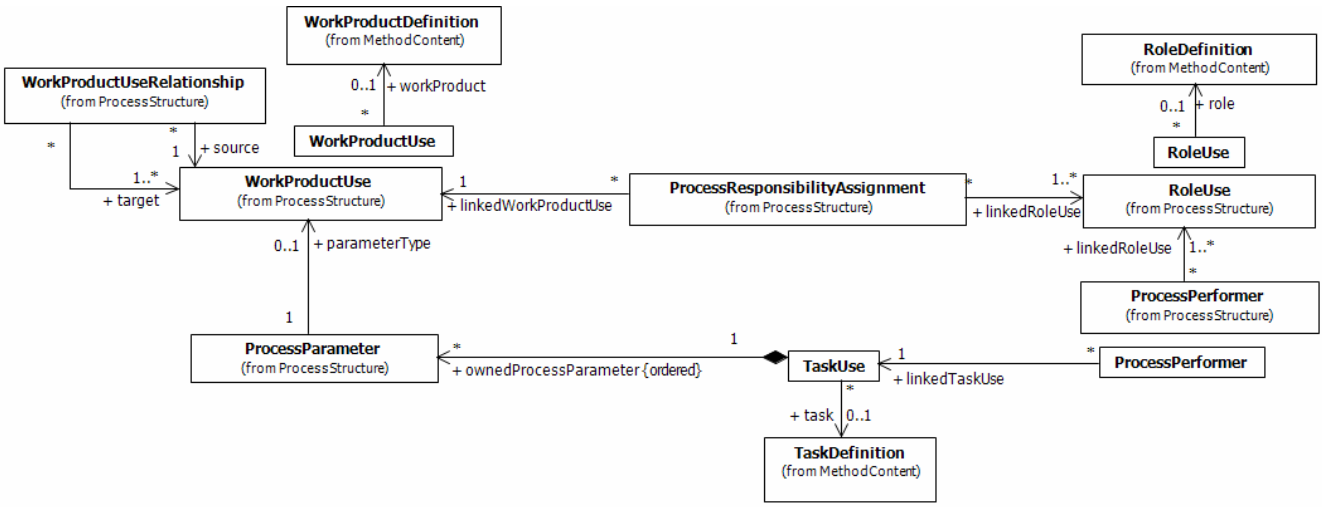


Figure 13.8 - Relationships of concrete Method Content Use classes

Description

A Method Content Use is an abstract generalization for special Breakdown Elements that references one concrete Method Content Element. A Method Content Use provides a proxy-like representation of a Method Content Element within breakdown structures. In addition to just referencing Method Content Elements, it allows overriding the Method Content Elements structural relationships by defining its own sets of associations and adding its own Content Description.

Semantics

Method Content Uses are the key concept for realizing the separation of processes from method content. A Method Content Use can be characterized as a reference object for one particular Method Content Element, which has its own relationships and properties. When a Method Content Use is created, it shall be provided with congruent copies of the relationships defined for the referenced content element. However, a user can modify these relationships for the particular process situation for which the Method Content Use has been created.

Figure 13.9 depicts a Task Definition example in which its performing Roles Definitions, as well as its input/output Work Product Definitions have been created. The situation of this example could be that the Task “Prioritize Use Cases” shall be performed differently in a project’s Inception phase than in its Elaboration phase (i.e., with different foci on different steps, utilizing different inputs, etc.). We see that the Task in Inception has an additional assisting role (Customer.Project Manager) and does not provide a relationship to the Risk List Work Product that had been defined as an optional input in the method content (i.e., steps of the Task that work with the Risk List will be omitted in this phase). The Method Content Use concept allows, in addition to defining new relationships, also defining specific process related properties. For example, Work Products can now express specific states they are in throughout the lifecycle. As we see in Figure 13.9, two different types of Use Case Models are distinguished here: a Use Case Model as it is normally being used during Inception, which describes use cases only briefly, versus use cases that have been detailed as it is the case during the Elaboration phase (note, that states have been annotated in the name for this example diagram, but that Work Product Use

provides formally defined tagged values for this information). Method Content Uses are not Content Elements and do not contain their own full descriptions. They rather refer back to their original Content Elements using the trace dependencies as depicted in the diagram and might define changes for the Content Element's descriptions, valid only for this particular occurrence of the Method Content Use. Note that for simplicity of the example diagram not all trace dependencies have been drawn, but that all other Method Content Uses also need to trace back to their respective method content counter parts.

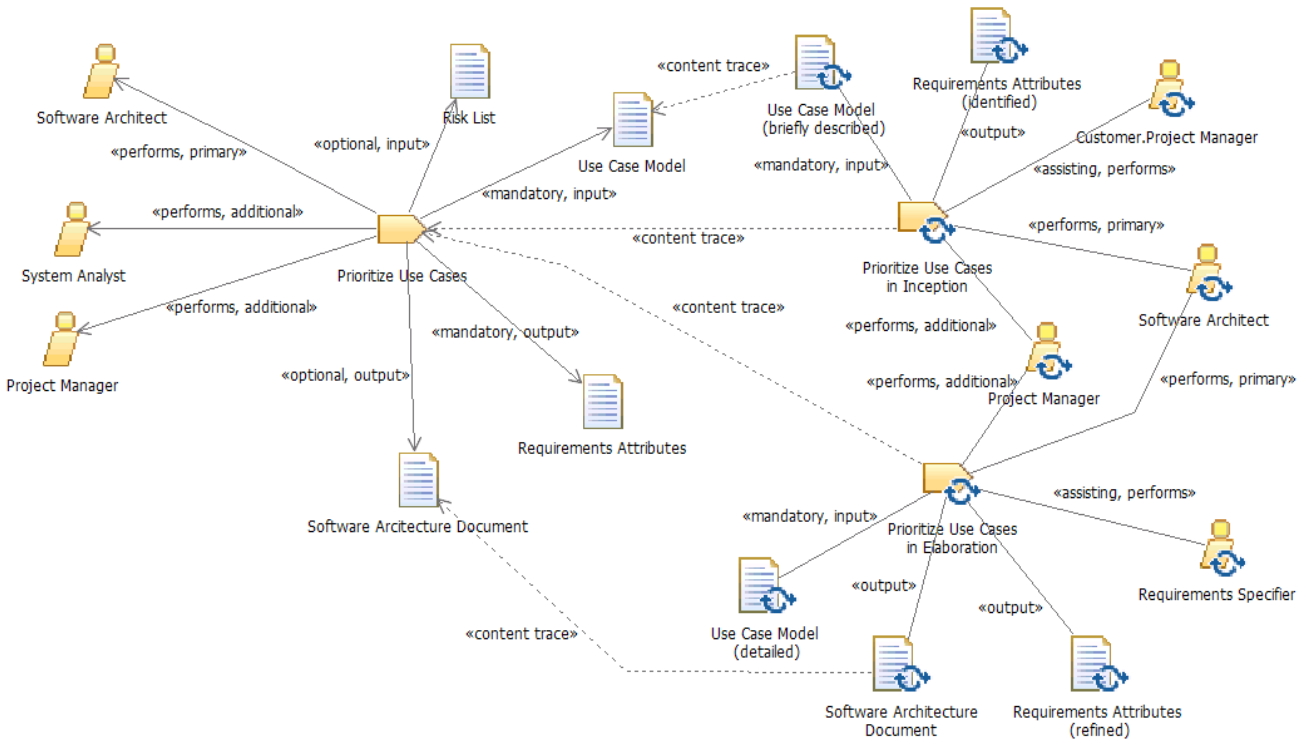
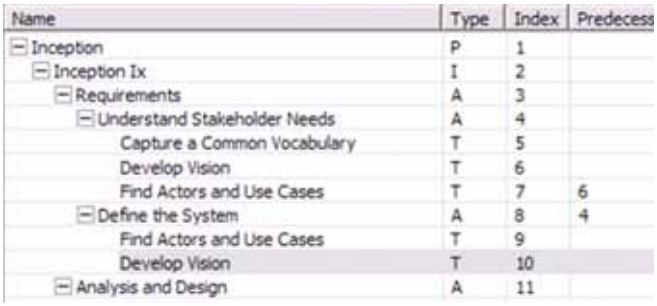


Figure 13.9 - Examples for Method Content Uses (right) referencing Method Content (left)

Note that Method Content Uses can be presented with UML diagrams using the SPEM 2.0 Profile as shown in Figure 13.9. However, a process definition for a typical development project will comprise many more Method Content Use instances and many more relationships instantiating many more relationship types such as predecessor association or presentation order associations. Therefore, a more practical presentation than diagrams allowing flexibly for filtering information is required. This practical presentation is depicted in Figure 13.10, showing a typical Work Breakdown Structure presentation as it is also being utilized by popular project planning tools. In this breakdown structure example, we see two Method Content Use instances for each of the Tasks "Develop Vision," "Find Actors and Use Cases." The only Breakdown Element associations shown in this example are the Nesting and Predecessor associations, though. One can see that the first "Find Actors and Use Cases" Task Use has a different predecessor than the second one (i.e., none defined for the second one, indicating that it can be performed in parallel to the "Develop Vision" Task). Nonetheless, other associations such as the inputs for the Task Uses can also be depicted in such breakdown structure presentations as it will be shown later in this specification with more examples (e.g., see Figure 9.16).

Figure 13.10 also illustrates that Method Content Uses are defined within the context of a specific Activity. For example, the figure shows the Method Content Use for the "Develop Vision" Task within the scope of the "Understand Stakeholder Needs" Activity as well as the "Define the System" Activity. Within the scope of each of these Activities, individual

Method Content Use relationships and properties will be defined. Thus, the “Develop Vision” Task can be performed by different Roles and with different input/output Work Products in the first Activity than the second Activity (as depicted in Figure 13.9 as well; however not showing the relationship of the Task Uses to Activities).



Name	Type	Index	Predecess
Inception	P	1	
Inception Ix	I	2	
Requirements	A	3	
Understand Stakeholder Needs	A	4	
Capture a Common Vocabulary	T	5	
Develop Vision	T	6	
Find Actors and Use Cases	T	7	6
Define the System	A	8	4
Find Actors and Use Cases	T	9	
Develop Vision	T	10	
Analysis and Design	A	11	

Figure 13.10 - Method Content Uses in a Work Breakdown Structure

The concrete Method Content Use classes defined in subsequent sections below have specific associations and attributes. Figure 13.8 provides an overview to all Method Content Use types defined in the Process package, their attributes and associations. We see that each Method Content Use traces back to the core content element they refer to. Method Content Uses also define additional associations than the associations defined among the core content elements. For example, External Inputs between Task Uses and Work Product Uses are not defined in Core Method Content, because it expresses information that is specific to a process and not available/known for method content.

SPEM 2.0 Profile Notation

No stereotype defined. See subclasses

13.8 Planning Data

Super Class

Process Element

Description

Planning Data is a Process Element that adds planning data to Breakdown Elements when it is used for generating project plans from a process.

Association Properties

startDate: String	Defines the absolute start date for the Breakdown Element.
finishDate: String	Defines the absolute finish date for the Breakdown Element.
rank: Integer	Defines the ranking of the Breakdown Element relative to other Breakdown Elements in same Activity.
duration: String	Duration of a Breakdown Element. Could be an exact recommended duration, or a ratio.

Semantics

Planning Data factors out specific optional data needed for populating plans.

SPEM 2.0 Profile Notation

The Planned Element stereotype can be used as a superclass for other stereotypes that need to store planning data such as Activity (Section 13.1) or Task Use (Section 13.14).

Stereotype	Metaclass	Keyword	Properties	Abstract	Icon
PlannedElement	n/a	n/a	startDate, finishDate, rank, duration	yes	n/a

13.9 Process Kind

Super Class

Kind (from Core)

Process Element

Description

Process Kind is a Kind and Process Element (see Figure 13.5) that represents a process specific refinement of the Kind class defined in Core (Section 8.2). Only Process Elements can be used for Process Kinds. It can be packaged in Process Packages (Section 13.10).

Semantics

See Kind (Section 8.2).

13.10 Process Package

Super Class

Package (from Constructs in UML 2 Infrastructure)

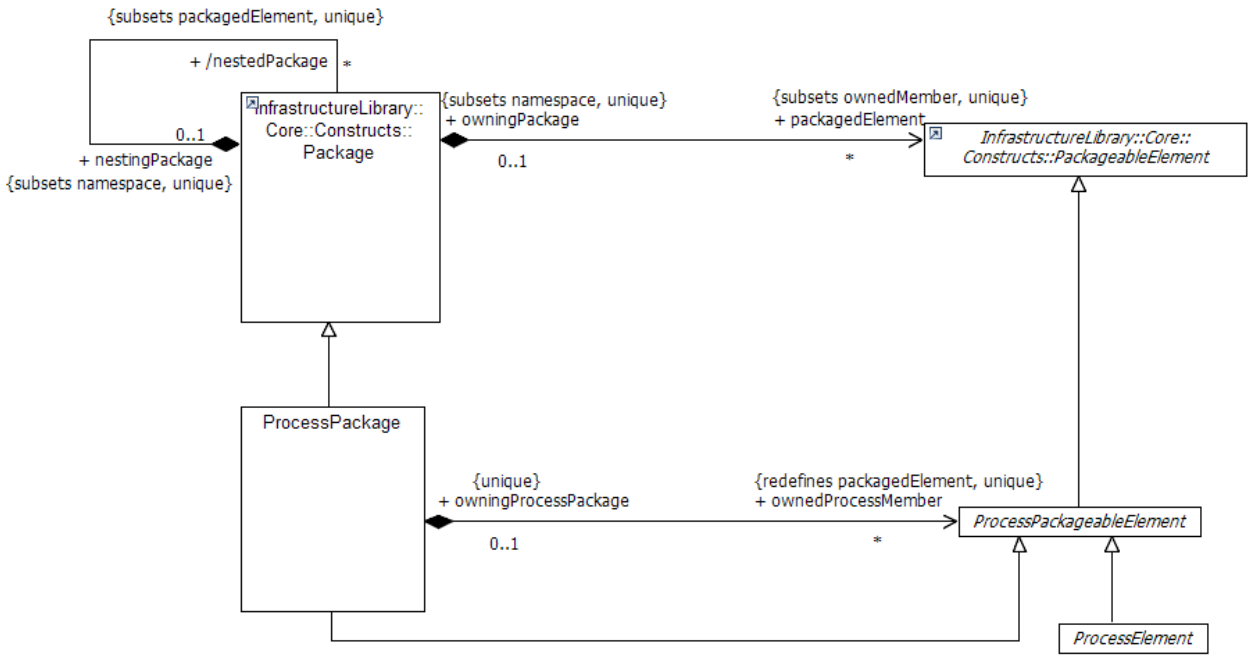


Figure 13.11 - Process Packages contains Process specific elements

Description

Process Package is a special Package that can only contain Process Packageable Elements. It redefines the packagedElement and ownedMembers association to only allow elements of these two types.

Association Properties

- ownedProcessElements: ProcessElement

  A Process Package can contain Process Elements that are used to define Breakdown and Processes. A Process Element instance can be part of only one Process Package instance.

Semantics

A process package can only contain Process Elements. A key separation of concerns in SPEM 2.0 is the distinction between Method Content and Process as outlined in Section 6.3.1. This separation is enforced by special package types, which do not allow the mixing of method content with processes. Method Content, introduced in Section 11.6, is managed in Method Content Packages.

SPEM 2.0 Profile Notation

Stereotype	Metaclass	Keyword	Properties	Abstract	Icon
ProcessPackage	Package	process package	n/a	no	

13.11 Process Packageable Element

Super Class

Packageable Element (from Constructs in UML 2 Infrastructure)

Description

Process Packageable Element is an abstract generalization for Process Package and Process Element supporting the redefinition of the packagedElement association of Process inherited from the UML 2 class.

Semantics

Process Packageable Element represents an element that can be packaged in a Process Package.

13.12 Process Performer

Super Class

Breakdown Element

Description

Process Performer in this meta-model package extends the Process Performer of the Process Structure meta-model package (Section 9.7) with an additional association to Task Use (see Figure 13.8). Process Performer can be used optionally to indicate performance of an Activity or a Task Use.

Association Properties

- linkedTaskUse: TaskUse — This association links exactly one Task Use to a Process Performer.

Semantics

See Process Performer in Process Structure (Section 9.7).

SPEM 2.0 Profile Notation

Use Work Definition Performer (see Section 8.6).

13.13 Role Use

Super Class

Method Content Use

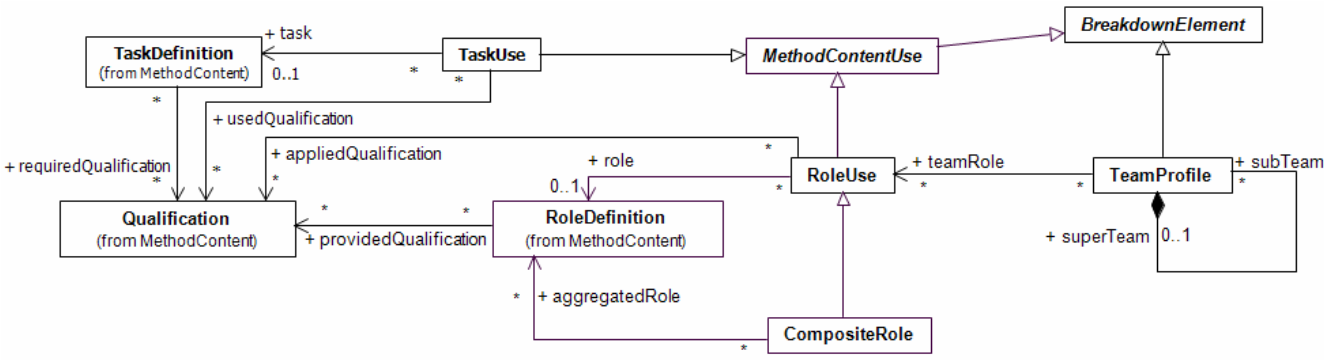


Figure 13.12 - A Team Profile consists of Role Uses and/or Composite Roles

Description

A Role Use represents a Role in the context of one specific Activity. Every breakdown structure can define different relationships of Role Uses to Task Uses and Work Product Uses. Therefore, one role can be represented by many Role Uses, each within the context of an Activity with its own set of relationships.

Association Properties

- role: RoleDefinition

  This association represents the reference from the Method Content Use to the method content element it refers to. Every Role Use can reference only one Role Definition. However, a Role Definition can be represented by many Role Uses. In the SPEM 2.0 Profile, this association is instantiated by a dependency relationship with a trace stereotype.

- appliedQualification: Qualification

  A Role Use can select a sub-set of valid Qualifications defined for the Role Definition for this one use of the Role Definition in the context of a particular Activity.

Semantics

Every breakdown structure can define different teams comprised of specific sets of Role Uses and Composite Roles. Within such a Team Profile, every Role Use can define different relationships as depicted in Figure 13.8.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
RoleUse	BreakdownElement / Classifier	role use	n/a	no	SPEM 1.1 backwards compatibility icon:
MethodContentTrace	/ Dependency	content trace	n/a	no	n/a

13.14 Task Use

Super Class

Method Content Use

Work Breakdown Element

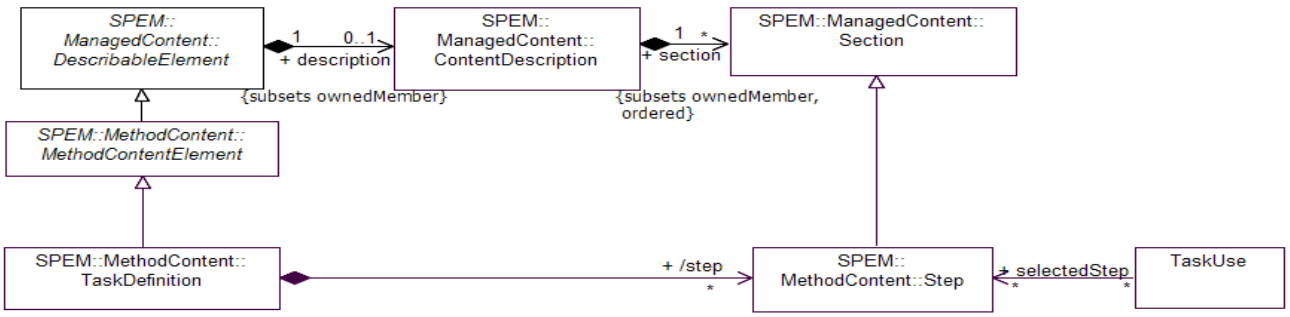


Figure 13.13 - Selecting Steps for a Task Use

Description

A Task Use is a Method Content Use and Work Breakdown Element that represents a proxy for a Task Definition in the context of one specific Activity. Every breakdown structure can define different relationships of Task Uses to Work Product Uses and Role Uses. Therefore, one Task Definition can be represented by many Task Uses each within the context of an Activity with its own set of relationships.

A key difference between Method Content and Process is that a Method Content Element, such as Task Definition, describes all aspects of doing work defined around this task. This description is managed in steps, which are modeled as Sections of the Task Definitions' Content Descriptions. When applying a Task Definition in a Process' Activity with a Task Use, a Process Engineer needs to indicate that at that particular point in time in the Process definition for which the Task Use has been created, only a subset of steps shall be performed. The engineer defines this selection using the selectedSteps association depicted in Figure 13.11. If s/he wants to add steps to a Task Use, s/he can describe these either pragmatically in the refinedDescription attribute, or 'properly' by creating a contributing Task Definition to the Task the Task Use refers to (see Section 14.6 on Contribution).

Association Properties

• task: TaskDefinition	This association represents the reference from the Method Content Use to the Method Content Element it refers to. Every Task Use can reference only one Task Definition. However, a Task Definition can be represented by many Task Uses. In the SPEM 2.0 Profile this association is instantiated by a dependency relationship with a trace stereotype.
• ownedProcessParameter: ProcessParamter	This ordered association links a Task Use to its Process Parameters.
• selectedStep: Step	List the steps that have been selected to be performed for the instance of the Task Use.
• usedQualification: Qualification	A Task Use can select a sub-set of valid Qualifications defined for the Task Definition for this one use of the Task Definition in the context of a particular Activity.

Semantics

See Method Content Use in Section 13.7 for details.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
TaskUse	WorkBreakdownElement, PlannedElement / Classifier, Action	task use	n/a	no	
MethodContentTrace	/ Dependency	content trace	n/a	no	n/a

13.15 Team Profile

Super Class

Breakdown Element

Description

A Team Profile is a Breakdown Element that groups Role Uses or Composite Roles defining a nested hierarchy of teams and team members.

Association Properties

- subTeam: TeamProfile — This composition association relates a Sub-Team to its Super-Team, i.e., the Team that it is part of.
- teamRole: RoleUse — This association specifies the roles of team members with Role Uses.

Semantics

Work assignments and Work Product responsibilities can be different from Activity to Activity in a development project. Different phases require different staffing profiles, i.e., different skills and resources doing different types of work. Therefore, a process needs to define such different profiles in a flexible manner. Whereas Core Method Content defines standard responsibilities and assignments, a process expressed in breakdown structures needs to be able to refine and redefine these throughout its definition. Role Uses, Composite Roles, as well as Team Profiles provide the data structure necessary to achieve this flexibility and to provide a process user with the capability to define different teams and role relationships for every Activity (including Activities on any nesting-level as well as Iterations or Phases).

Hence, in addition to the work breakdown and work product breakdown structures defined so far, Team Profiles are used to define a third type of breakdown structure: team breakdown structures. These are created as an activity-specific hierarchy of Team Profiles comprised of Role Uses and Composite Roles. These structures can be presented as well-known Organizational Charts as depicted in Figure 13.14. The diagram shows three instances of Team Profile forming a hierarchy (the top-level node being just a logical node grouping all the teams and roles defined for an Activity together). Every Team Profile instance has a number of Roles associated, which will be represented as Role Uses.

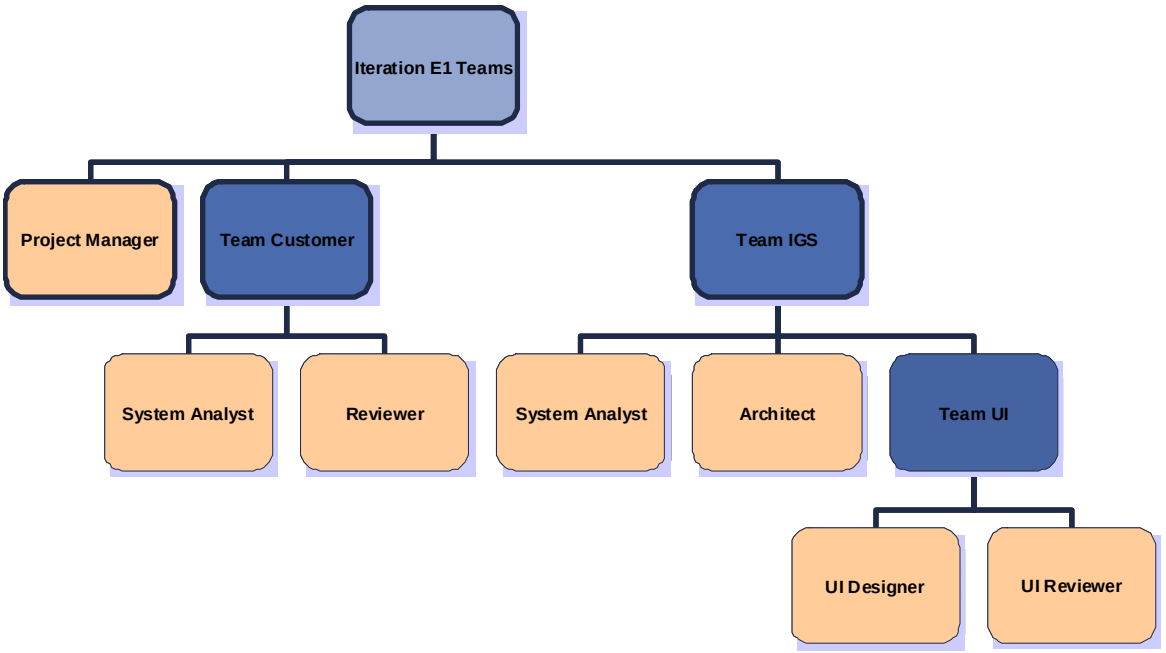


Figure 13.14 - Example of a Team Profile with nested Team Profiles and Role Uses

Just as with any other Breakdown Element and Method Content Uses, Team Profiles can be defined within the scope of any Activity in a breakdown structure. In other words, every Activity can define its own Team Profiles consisting of activity-specific Role Uses and Composite Roles that are reusable for lower level Activities. Typically, Team Profiles are defined on the level of Iterations (as depicted in Figure 13.15), Phases or other higher-level Activity.

	Team Breakdown Structure	Team	Model Info
1	− DeliveryProcess.RUP for Small Projects		
2	− Developer		Composite Role
3	Implementer		
4	Tester		
5	Phase.Inception		
6	− Phase.Elaboration		
7	− Iteration.E1		
8	Project Manager		local
9	− Team Customer		
10	System Analyst	Team Customer	
11	Reviewer	Team Customer	
12	− Team IGS		
13	System Analyst	Team IGS	
14	Architect	Team IGS	
15	− Subteam IGS.UI	Team IGS	
16	UI Designer	Team IGS.UI	
17	UI Reviewer	Team IGS.UI	
18	− Discipline.Requirements		
19	− Activity.Understand Stakeholder Needs		
20	Architect	Team IGS	no change (from Iteration.E1)
21	System Analyst	Team Customer	no change (from Iteration.E.1)
22	System Analyst	Team IGS	changed relationships (from Iteration.E1)
23	UI Designer	Team IGS.UI	augmented description (from Iteration.E1)
24	Project Manager		no change (from Iteration.E1)
25	Requirements Specifier		local
26	Developer		Composite Role (from RUP for Small Projects)
27	Activity.Define the System		
28	Activity.Manage the Scope		
29	Activity.Refine the System Definition		
30	Phase.Construction		
31	Phase.Transition		

Figure 13.15 - Example for Team Profiles used in a Team Breakdown Structure

Figure 13.15 provides an example of a Team Breakdown Structure that applies the Team Profiles from Figure 13.14 locally to an Iteration called ‘Iteration E.1.’ It defines the Team Profiles in Rows 8 to 17 and then applies these in sub-Activities. For example, the Activity ‘Understand Stakeholder Needs’ utilizes several roles from the different teams, e.g., an Architect from the IGS team, the System Analysts from both teams, etc. A project manager using this process for planning can directly see in this Team Breakdown Structure Activity for Activity which roles s/he has to find resources for. Every occurrence of a Team Profile Role is represented by another Role Use which is used for relationships to Task Uses and Work Product Uses relating to the other two breakdown structures. The rules for matching the Method Content Uses from a lower Activity such as ‘Understand Stakeholder Needs’ to a higher level Activity such as ‘Iteration.E1’ are straight-forward:

- If a Role Use links to the same Role Definition (or contributed/replaced Role Definition; see Chapter 14, Method-Plugins for more details) in Method Content, then a Role Use in a parent Activity and both Role Uses are linked to the same Team Profile, then the two Role Uses represent the same role.
  (Examples from Figure 13.15 are Architect, UI Designer, etc.)
- If the two Role Uses relate to different teams, two separate Roles are represented.
  (Examples from Figure 13.15 are the two System Analysts, which represent two different Roles.)
- If the Role Use links to a Role that is not part of any Team or listed in a higher level Activity, then the Role Use is defined an Activity local Role, which can be ‘reused’ for lower level Activities.
  (Example from Figure 13.15 are the Project Manager, which is defined as local in Activity ‘Iteration.E1’ and then reused in Activity ‘Understand Stakeholder Needs,’ or the Requirements Specifier that is locally defined on the lowest Activity level.)

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
TeamProfile	BreakdownElement / Classifier	team profile	n/a	no	

13.16 Work Product Use

Super Class

Method Content Use

Description

A Work Product Use represents a Work Product Definition in the context of one specific Activity. Every breakdown structure can define different relationships of Work Product Uses to Task Uses and Role Uses. Therefore, one Work Product Definition can be represented by many Work Product Uses each within the context of an Activity with its own set of relationships.

Association Properties

• workProduct: WorkProductDefinition	This association represents the reference from the Method Content Use to the core content element it refers to. Every Work Product Use can reference only one Work Product Definition. However, a Work Product can be represented by many Work Product Uses. In the SPEM 2.0 Profile this association is instantiated by a dependency relationship with a trace stereotype.

Semantics

See Method Content Use in Section 13.7 for details.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
WorkProductUse	BreakdownElement / Classifier	work product use	n/a	no	
MethodContentTrace	/ Dependency	content trace	n/a	no	n/a

14 Method Plugin

The Method Plugin package defines capabilities around managing whole libraries of Method Content and Processes. It addresses the concern of scaling to large Method Libraries by defining Method Plugins and Method Configurations.

The Method-Plugin package defines extensibility and variability mechanisms for method content and processes. It provides more flexibility in defining different variants of method content and processes by allowing content and process fragments to be plugged-in on demand, thus creating tailored or specialized content only when it is required and which can be maintained as separate units worked on by distributed teams.

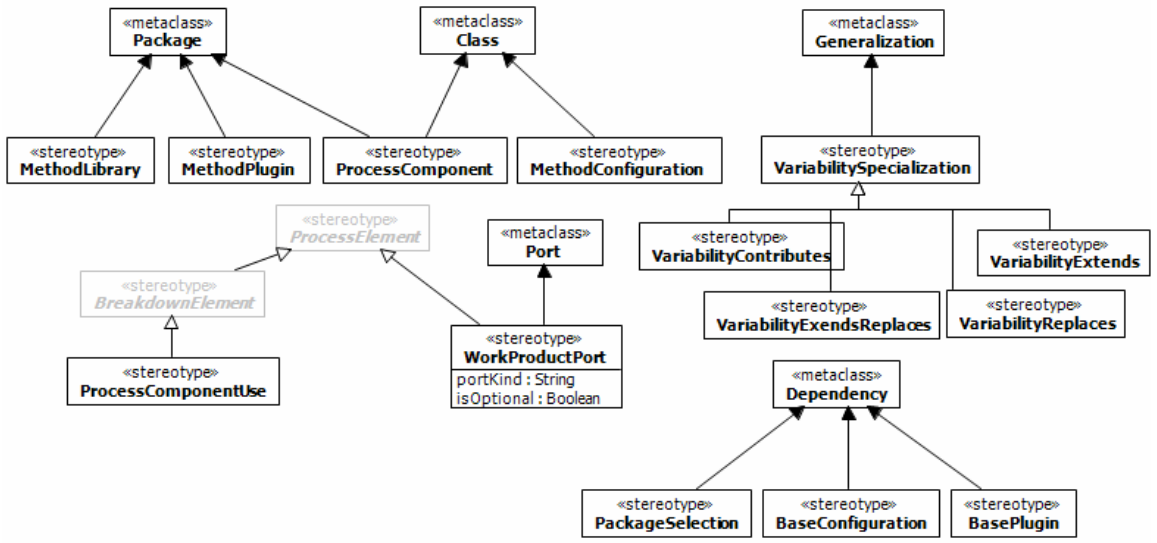


Figure 14.1 - The SPEM 2.0 UML 2 Profile stereotypes defined in the Method Plugin package

Methods and Processes are never developed in isolation. Many concurrent teams of process engineers might work on different or the same parts of a Method Content or particular Processes. They have to either coordinate their work or preferably to modularize Method Content and Process definitions that they can work on in isolation without any shared resource problems.

Moreover, work on a development Process is never final because they have to be tailored for every specific project situation. To avoid developing Delivery Processes from scratch for every specific project situation, a truly reusable Process allows defining many different variations on it. Therefore, tailoring and extension of pre-defined processes is a key requirement for organizations adopting and implementing processes. For example, imagine a scenario in which a development organization purchases a third-party Delivery Process from a method vendor and its underlying method content. While implementing the process in their organization, they might discover that they would like to tailor the process to their particular needs and organization specific variations. If they would directly modify the purchased process, they would mingle their additions and changes with the original process. If a new version of the original process is released by the vendor, it would be very hard for them to find and reproduce these changes to the new process version.

The package Method Plugin adds the capabilities of modularization and extensibility of Method Content and Processes to SPEM 2.0. In other words, it supports tailoring of Method Content and Processes without directly modifying them, but by describing changes from a separate unit called a Method Plug-in. This new capability is achieved by adding new concepts to the meta-model:

A. All SPEM Elements are organized in Method Plugins. Method Plugins are containers for Content and Process Packages. Every SPEM element instance is part of exactly one Method Plugin, i.e., Method Plugins partition the space of all elements.

B. Method Plugins can define extensions to other Method Plugins. One Method Plugin can define extensions for many other Plugins and can be extended by many Plugins.

C. The Method Plugin meta-model package adds a new set of relationships and semantics to specific SPEM Elements that define how these SPEM Elements can extend each other across the boundary of Method Plugins or Method Packages (to allow modularization even within one Method Plugin).

Method Configurations define visibility spaces or working sets for Method Content and Process, allowing a Process Engineer to focus on relevant parts of Library instead of seeing the whole library all at once. By defining Configurations as subsets of Plugins and Method Packages that shall be considered for process authoring only, Configurations also define the space of variability interpretation to avoid conflicts among 'competing' content (e.g., content that replaces the same elements).

The Library Configuration classifier adds additional capabilities for packaging Method Plugins and scaling method and process authoring work. It defines the key concept of Library that comprises Method Plugins and Method Configurations. A Library represents the overall physical container for all SPEM Elements. Libraries represent a separate Method 'universe,' i.e., a Method Library cannot reference elements from another Method Library. A Configuration is a logical subset of Plugins and Method Packages, i.e., selections of Method Packages. They are used to define working sets for process authoring, providing a process engineer with lists of elements to be used for creating process within such a Configuration. Configurations also define the space of variability interpretation to avoid conflicts among 'competing' content (e.g., more than one variability element that replaces the same element) as well as define sets of Method Content and Processes for export and distribution.

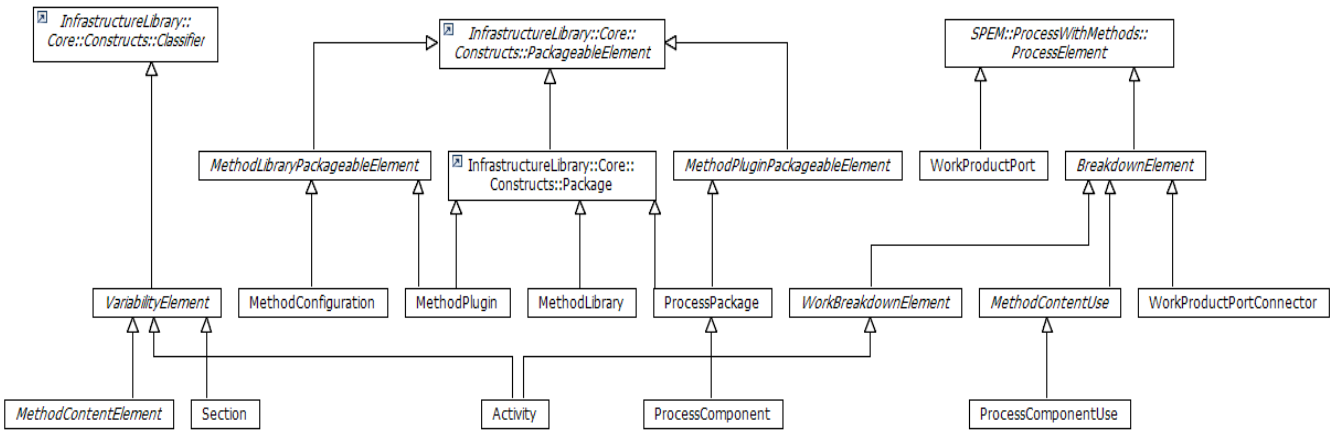


Figure 14.2 - Taxonomy of the classes defined in Method Plugin meta-model package

14.1 Activity

Super Class

Variability Element

Description

Activity in the package Method Plugin inherits from Variability Element to extend Activity with new capabilities for variability. It extends the class with association relating a Process to one default and many optional valid Configurations (see Figure 14.3).

Association Properties

- defaultContext: MethodConfiguration — An Activity that represents a process can have one default configuration defining the visibility space of the activity's sub-elements. Every child activity will automatically inherit its parent's default configuration unless it defines its own overriding configuration.
- validContexts: MethodConfiguration — An Activity that represents a process can have many additional configurations that have been verified to also produce valid results. Process Elements (e.g., Method Content Uses) that refer to content packages that are defined outside the scope of such a configuration would not be shown in the process when published or used under such a configuration. This allows easily removing content from a process by removing content packages from the related configuration.

Semantics

Activity inherits the semantics of Variability Element, which provides key mechanism to enable dynamic modification of Activities in a process from a Method Plugin. The relationships to configurations describe valid contexts for the Process within a Method Library indicating under which Configurations an Activity is well defined.

14.2 Method Configuration

Super Class

Classifier (from UML 2 Infrastructure)

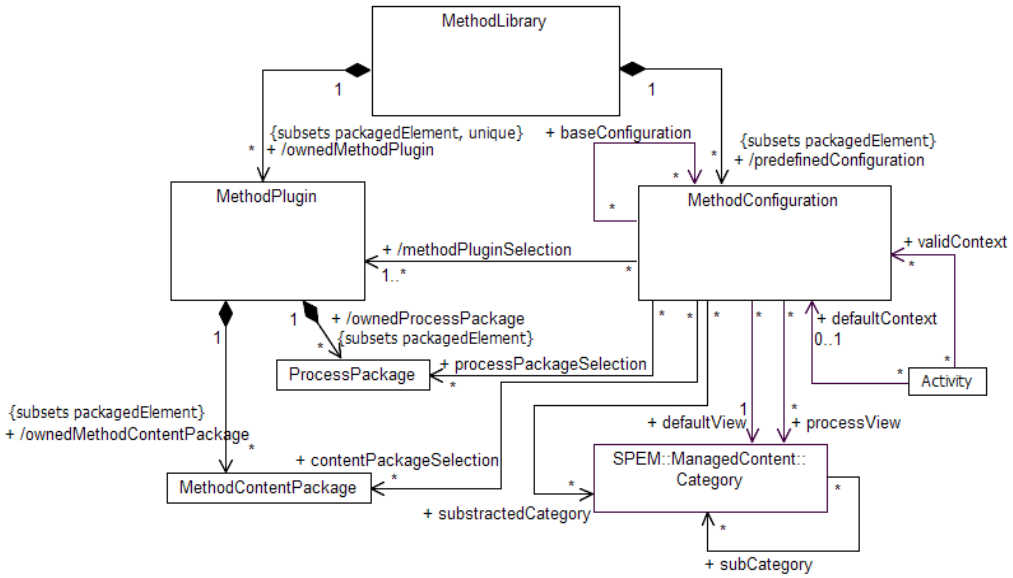


Figure 14.3 - Definition of Method Library and Method Configurations

Description

A Method Configuration is a collection of selected Method Plugins, as well as subsets of Method Content Packages and Process Packages of respective Method Plugins. The definition of a configuration can be further refined by subtraction of all Content Elements categorized by a Category associated via the subtracted Category association.

Association Properties

- /methodPluginSelection: MethodPlugin — This derived association includes all Method Plugins that own packages that have been selected as part of the Method configuration using the methodPackageSelection association.
- contentPackageSelection: ContentPackage — A selection of packages to be included in the configuration.
- processPackageSelection: ProcessPackage — A selection of packages to be included in the configuration.

- baseConfigurations: MethodConfiguration

The definition of a configuration can be based on the definitions of other configurations. For example, a configuration A could be defined as the superset of configurations B, C, and D plus additional Plugins and Packages. In such a case, A would list B, C, and D as its baseConfigurations. If any of these base configurations changes, all updates would be automatically valid for A as well, thus reducing maintainability overhead for overlapping/dependent configurations.

Semantics

A Method Configuration defines a logical subset on a Method Library. It is defined by selection of Method Packages. Figure 14.4 depicts an example of a Method Configuration by shading the packages that have been selected for a Method Configuration in red. As a result, the Configuration of Figure 14.4 consists of the plug-ins 'Base Model,' 'RUP Software Family,' 'Framework RUP,' and 'COTS Plug-in' as well as only the packages of those plug-ins that have been selected (i.e., the red ones).

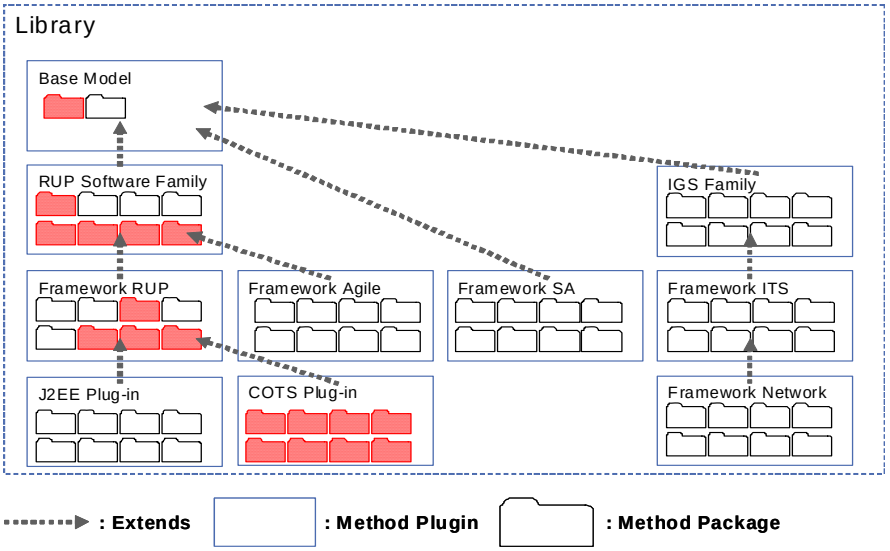


Figure 14.4 - Illustration of a Method Configuration: all colored packages a part of a Configuration

A Method Configuration defines a visibility space within a Method Library that can be used to filter Method Content and Processes. An Activity representing a Process can be valid for many different Method Configurations (see association validContext for Activity in Section 14.1), but each Configuration includes or excludes particular content for specific situations. For example, a Process can be defined to include content for developing schemas for different types of database management systems such as content for RDBMS and OODBMS. When using such a Process, a user might want to select just one type of DBMS support for the project. Hence, s/he wants to exclude all content for the alternative development methods for the other type of DBMS. S/he achieves this by defining a configuration that excludes the respective content; in other words she removes this unwanted content from the current visibility space defined by the configuration. An implementation of this specification would interpret the absence of specific Method Content Packages from a selected Method Configuration, which contain elements referred to by the Method Content Uses in the Process Definition, in the way that these Method Content Uses would also not be visible, i.e., ignored by the Process presentation. Using this configuration technique, a Process can be defined to include many alternative development techniques, but

ultimately the configurations would decide which techniques actually would be used. A Process maintains lists of valid configurations in the validContext association, i.e., configurations that have been “tested” by the process author to prove a valid and useful process variant.

Another application of Method configuration is to create new Libraries by exporting all contents of a configuration into its own new standalone Library. A prerequisite for such an export would be that the configuration needs to include the full transitive closure of all elements all other elements depend on. Such a configuration-based export functionality is useful for selling and shipping only parts of a Method Library to customers.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
MethodConfiguration	/ Class	configuration	n/a	no	n/a
PackageSelection	/ Dependency	package selection	n/a	no	n/a
BaseConfiguration	/ Dependency	base configuration	n/a	no	n/a

14.3 Method Library

Super Class

Package (from UML 2 Infrastructure)

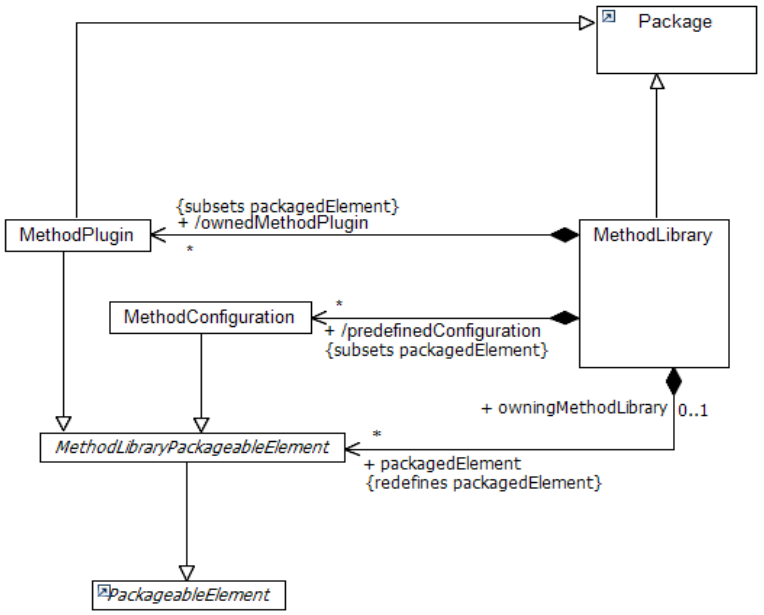


Figure 14.5 - A Method Library is a container for Plugins and Configurations

Description

A Method Library is a physical container for Method Plugins and Method Configuration definitions. All SPEM 2.0 elements are stored in a Method Library.

Association Properties

• /ownedMethodPlugins: MethodPlugin	A Method Library physically comprised of Method Plugins. A Method Plugin can only be part of one Method Library. This property subsets the redefined packagedElement.
• /predefinedConfigurations: MethodConfiguration	A Method Library stores a set of predefined Method Configurations that are regularly used by the library authors (e.g., the configurations that they ship to their users). This property subsets the redefined packagedElement.
• packagedElement: MethodLibraryPackage-ableElement	This association redefines the association from UML Infrastructure Package to only allow MethodLibraryPackageableElements.

Semantics

No additional semantics defined.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
MethodLibrary	/ Package	library	n/a	no	

14.4 Method Library Packageable Element

Super Class

Packageable Element (from Constructs in UML 2 Infrastructure)

Description

Method Library Element is an abstract generalization for Method Plugin and Method Configuration supporting the redefinition of the packagedElement association of Method Library inherited from the UML 2 class.

Semantics

Method Library Packageable Element represents an element that can be packaged in a Method Library. It prevents SPEM 2.0 user from nesting Method Libraries.

14.5 Method Plugin

Super Class

Package (from UML2 Infrastructure Library)

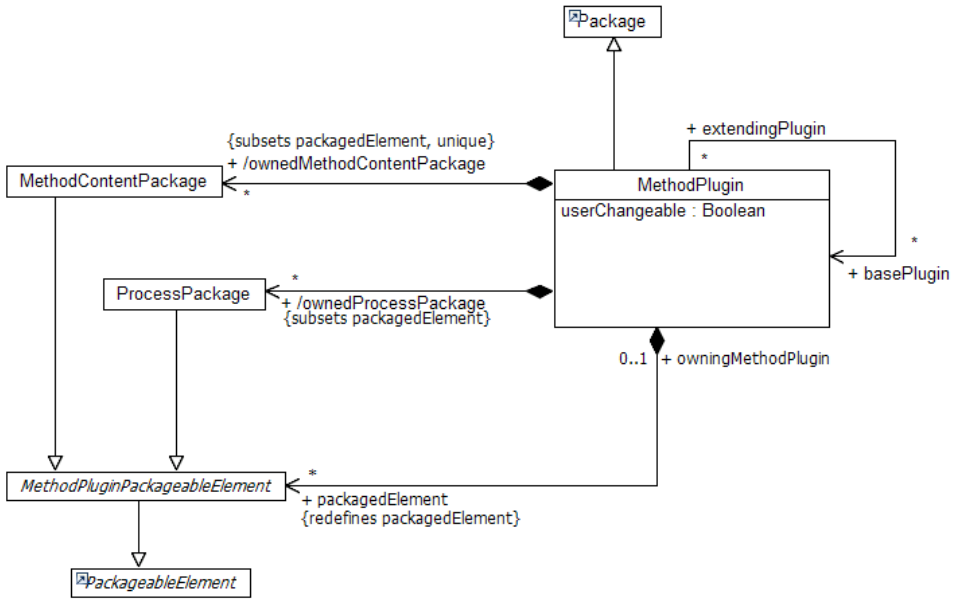


Figure 14.6 - Method Plugins are containers for Method Packages

Description

A Method Plugin is a Package that represents a physical container for Content and Process Packages. It defines a granularity level for the modularization and organization of method content and processes. A Method Plugin can extend many other Method Plugins and it can be extended by many Method Plugins. It can also be used stand-alone with no Extension relationship to other plug-ins.

Association Properties

• /ownedMethodContentPackage: Method-ContentPackage	This composition association represents that every Content Package is part of exactly one Method Plugin. This property subsets the redefined packagedElement.
• /ownedProcessPackage: ProcessPackage	This composition association represents that every Process Package is part of exactly one Method Plugin. This property subsets the redefined packagedElement.
• packagedElement: MethodPlugin PackageableElement	This association redefines the association from UML Infrastructure Package to only allow MethodPluginPackageableElements.

- basePlugin: MethodPlugin

This association defines that Method Plugins could extend many other Method Plugins. The extending plug-in contains new SPEM Elements, extends existing SPEM Elements and processes using Variability (e.g., contribution, replace, etc.) provides additional Processes, etc.

Semantics

Method Plugin conceptually represents a physical storage unit for configuration, modularization, extension, packaging, and deployment of method content and processes. A Process Architect would design his Plugins and allocate his content to these Plugins with requirements for extensibility, modularity, reuse, and maintainability in mind.

Figure 14.7 shows an example for a Method Plugin called “Rational Unified Process Framework.” The diagram shows samples from the Plugin’s contents as well as two other Plugins extending it. As most Method Plugins, this Plugin is structured into two main packages separating method content from processes: A Content Package with a hierarchy of sub-Content Packages as well as a Process Package containing the Plugin’s Processes. The Content Packages contain typical Content Elements such as Task, Roles, Work Products, and their relationships. The Process Package contains different Kinds of Processes.

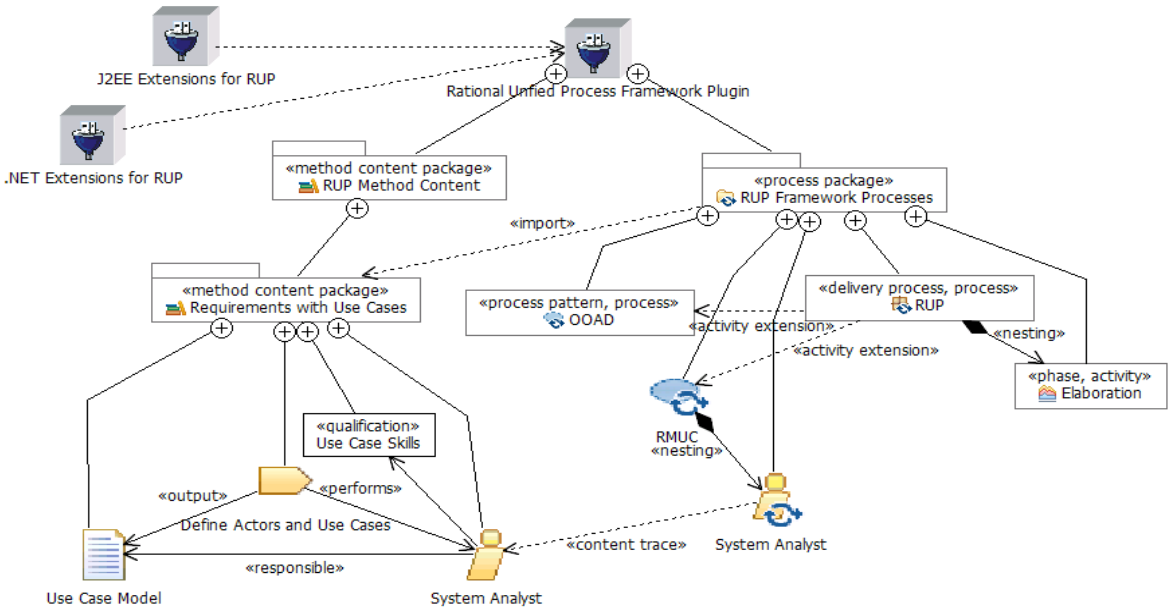


Figure 14.7 - Example Method Plugin with sample contents and its relationships

In addition to showing the typical structure of a Method Plugin, Figure 14.7 also depicts that two other Plugins have been defined that extend the RUP Framework Plugin with additional capabilities. Specific context-sensitive content has been factored out into separate Method Plugins; in this case content, which is dependent on the concrete development technology to be considered for a development process. This factoring allows one to alternatively choose one Plugin over the other, depending whether design for the J2EE or .NET platform is required.

Special extensibility mechanisms defined for the meta-classes Variability Element (Section 14.10) allow Method Plugin content to directly contribute new content, replace existing content, or to cross-reference to any Content Element or Process within another Plugin that it extends. Similar to UML 2’s ‘package merge’ mechanism transformation

interpretations, interpreting these Method Plugin mechanisms results into new extended Method Content and Processes. For example, the J2EE Extensions Plugin depicted in Figure 14.7 might contain additional steps for Tasks identifying design elements, new Work Products, extensions to existing Roles to be responsible for the new Work Products, additional relationships of existing Content Elements to new J2EE specific Guidance elements (such as Guidelines, White Papers, Checklists), additional Activities for a Delivery Process, new Process Patterns, etc. The Method Plugin defines these extensions using Variability Element relationships, and interpretation of these leads to new Method Content and Processes. See Section 14.6 and Section 14.11 for more details.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
MethodPlugin	/ Package	method plugin	n/a	no	
BasePlugin	/ Dependency	base plugin	n/a	no	n/a

14.6 Method Plugin Packageable Element

Super Class

Packageable Element (from Constructs in UML 2 Infrastructure)

Description

Method Plugin Element is an abstract generalization for Method Content Package and Process Package supporting the redefinition of the packagedElement association of Method Plugin inherited from the UML 2 class.

Semantics

Method Plugin Packageable Element represents an element that can be packaged in a Method Plugin. It prevents SPEM 2.0 user from nesting Method Plugins.

14.7 Process Component

Super Class

Process Package

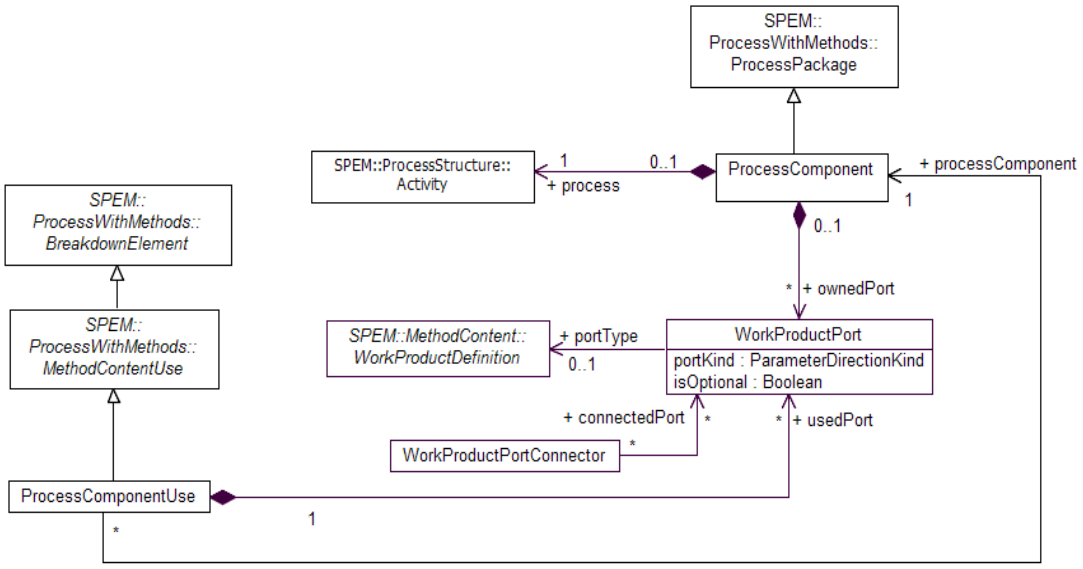


Figure 14.8 - Specification of Process Components and their relationships

Description

A Process Component is a special Process Package that applies the principles of encapsulation. A Process Component contains exactly one Process represented by an Activity, and defines a set of Work Product Ports that define the inputs and outputs for a Process Component. There might be many components defining the same Work Product Ports, but using different techniques (i.e., Process) to achieve similar outputs for similar inputs. Whereas the Work Product Port represents the component specifications (black box view on the component), good candidates for component realizations can be found in Process Patterns (a Process Kind for white box descriptions for the component, see Section 18.4.2).

Association Properties

- process: Activity — A Process Component contains exactly one Process that is physically encapsulated by the component.
- ownedPort: WorkProductPort — This association defines the ports required or provided by the Process Component. They define work product types used, produced, or changed by the process component.

Semantics

SPEM 2.0 supports replaceable and reusable Process Components realizing the principles of encapsulation. Certain situations in a software development project might require that concrete realizations of parts of the process remain undecided or will be decided by the executing team itself (e.g., in outsourcing situations). Process components support an assembly mechanism that is based on the “black box” principle. At any point during a project, the component “realization” detailing the work can be added to the process. The component approach also allows that different styles or techniques of doing work can be replaced with one another. For example, a software code output of a component could be produced with a model-driven development or a code-centric technique. The component concept encapsulates the actual work and lets the development team choose the appropriate technique and fill the component’s realization with their choice of Activities that produce the required outputs.

Assembling Process components is used typically in subcontracting context, where different processes and approaches are adopted by the subcontractors, and where the assembler process component is only focusing on the inputs and outputs of each subcontractor’s process component, namely the required and provided deliverables.

Figure 14.9 - Graphical Representation of a Process Component

Figure 14.9 shows an example for a graphical UML 2-based representation using stereotypes. It shows a Process Component named ‘Development’ that defines two input Work Product Ports ‘Analysis Model’ and ‘Design Model’ as well as an output port ‘Application.’

Assembling process components starts simply by connecting process components, through their input and output work product ports. That seems to be trivial, but there are several problems to solve:

- The number and types of input and output work products may not correspond within a set of process components.
- The invariant required for connected inputs and outputs may be contradictory.
- The names and constitution of work products may vary from one process component to another.
- The name and definition of roles may vary from one process component to another.

There is no guarantee that different process components will match, in particular if process components are complex, and originate from sources having different development approaches, the matching can be impossible. However, one can follow the following guidance for assembling and mapping different process components, in order to adjust the variations that can exist between process components:

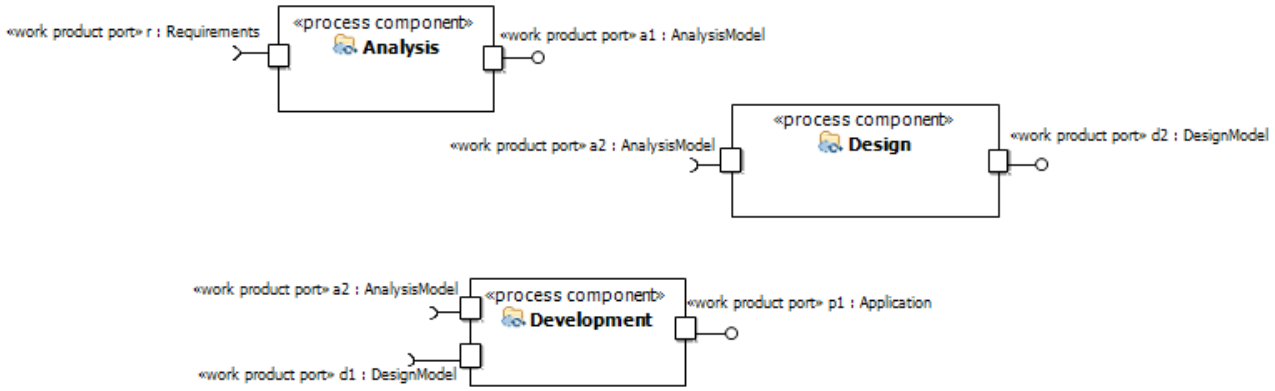


Figure 14.10 - Three Process Components to be assembled

An assembly mechanism similar to the UML2.0 notion of ports is used for that purpose. Connectors are lines between the various ports, in order to express the connections that one wants to establish between the process components. Optional input or output Work Products Ports do not need to be assembled, whereas it is mandatory to assemble every non-optional input or output Work Product Port.

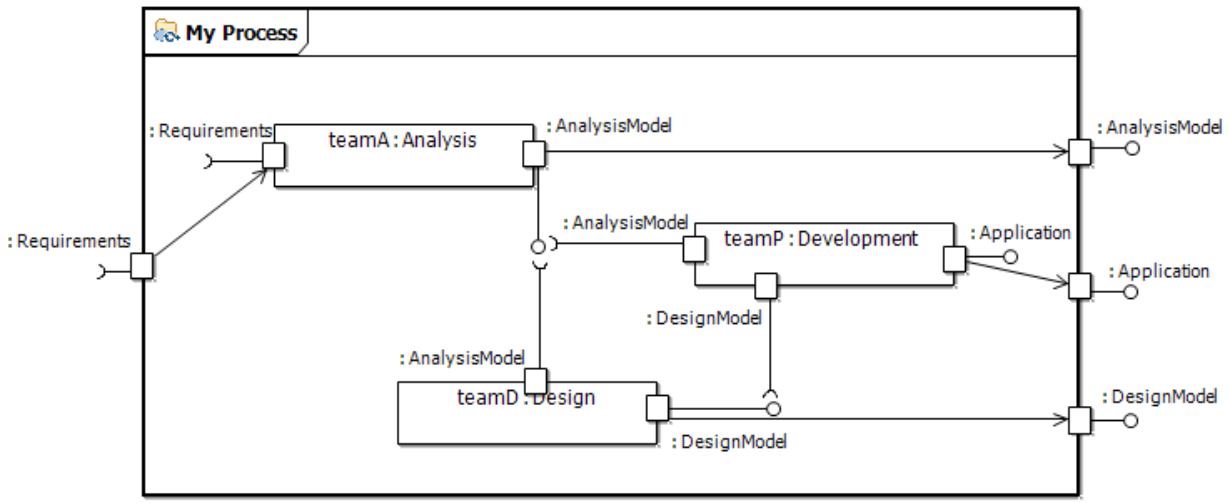


Figure 14.11 - Assembling Process Components

The assembly depicted in Figure 14.11 can also be established between Process Components: one Process Component can assemble several other Process Components. Once assembled, a Process can reuse the elements of the Process Components, such as the phases defined by the process components.

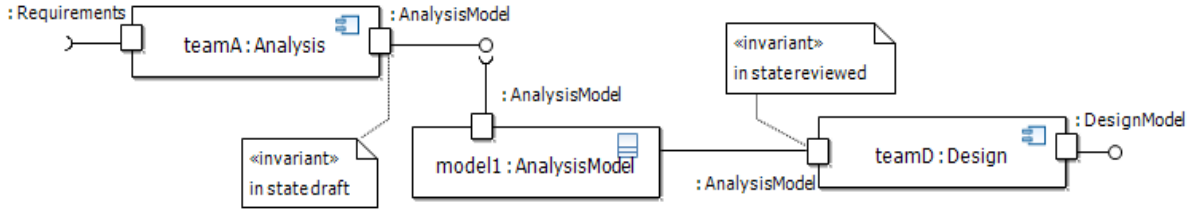


Figure 14.12 - Introduction of an intermediary Work Product in the assembly process

The assembly of process components may not be as immediate as depicted in Figure 14.11. There can exist name incompatibilities, and unresolved correspondences that will interfere with the assembly. In that situation, extra modeling efforts are necessary to adapt the existing elements of the assembled pieces. In particular, the process engineer can define additional activities within redefined phases, or add the missing work products within the assembling PC or process.

When input or output work product invariants do not match, such as when the provided state of an output work product does not match the required state of a connected input work product, there needs to be additional modeling precisions provided by the model of the assembler process. This can be solved by expressing that the assembling process does actions on the work product issued by a process component before passing it to the consumer process component as shown in Figure 14.12. This states that the coordination of the phases provided by the assembled process components needs some additional activities within the assembler process.

Another typical case is where no existing process component can provide a required (input) work product of an assembled process component. In that case, it is up to the assembler process to provide the missing element (see the case of “technical requirements” in Figure 14.13). This states that there will be activities within the assembler process that will provide that element.

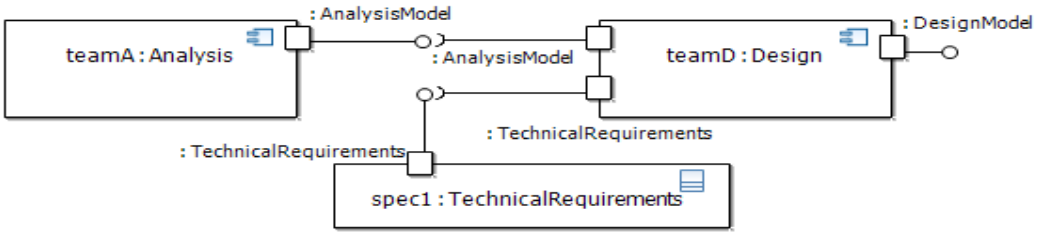


Figure 14.13 - Introduction of a missing input Work Product

Problems become more important when there is no defined correspondence between elements that need to be correlated. This happens typically when assembled work products have different names, and no established correspondence. This happens also when the name of roles changes between different process components, even if they correspond to each other. The correspondence between elements has to then be stated explicitly in the model by the process engineer, using Extends generalization links between these elements Figure 14.14.

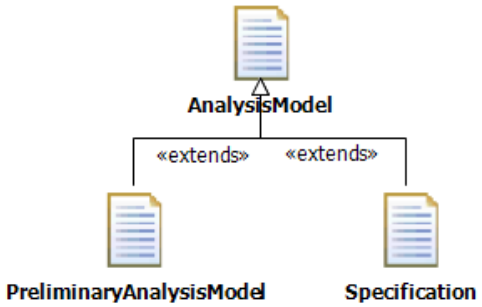


Figure 14.14 - Defining correspondences between different model elements

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
ProcessComponent	/ Package, Class	process component	n/a	no	SPEM 1.1 backwards compatibility icon:

14.8 Process Component Use

Super Class

Method Content Use

Description

A Process Component Use represents a Process Component application in any other Process defined by a breakdown structure. In other words, it represents a reference of the Process Component from within another Process. The Process Component Use is also the representation of the encapsulated Process in the context of another Process that applies it, hiding the details of the component realization in a breakdown structure. The Process Component Use can define its own set of relationships such as it own predecessors and successors as well as its own Ports that can be connected within the breakdown structure in which it is defined.

Association Properties

- processComponent: ProcessComponent — This association represents the reference from the Method Content Use to the Process Component it refers to. A Process Component Use represents exactly one Process Component.
- usedPort: WorkProductPort — This association links the Process Component Use to used Work Product Ports of the Process Component.

SPEM 2.0 Profile Notation

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
ProcessComponentUse	BreakdownElement / Classifier	process component use	n/a	no	n/a

14.9 Section

Super Class

Variability Element

Semantics

Section in the package Method Plugin inherits from Variability Element and extends Section defined in Managed Content (Section 11.6) with new capabilities for variability.

For example, when a Task Definition contributes to another Task Definition its description association is contributed including its Sections (i.e., its Steps), which are modeled as parts of the Content Description instance. Sections can be nested and therefore the Task Definition's descriptions can be flexibly organized in Steps with sub-Steps. Sections are Variability Elements themselves, so they can contribute to each other. For example, one could model a Task Definition Step as a Section instance without relating it to a Task Definition's Content Description that directly contributes to (or replaces) another Section that is part of a Content Description. This contribution (or replacement) would add new description text to the original step description (or replace the original step description). Another example would be to contribute to Guidance; for example, contribute new check list items organized as Sections to an existing check list guidance element.

Association Properties

- predecessor: Section — This association defines the predecessor for contributed Sections to be inserted into an existing list of Sections of a Content Description.

Semantics

Section inherits the semantics of Variability Element.

14.10 Variability Element

Super Class

Classifier (from Constructs in UML 2 Infrastructure)

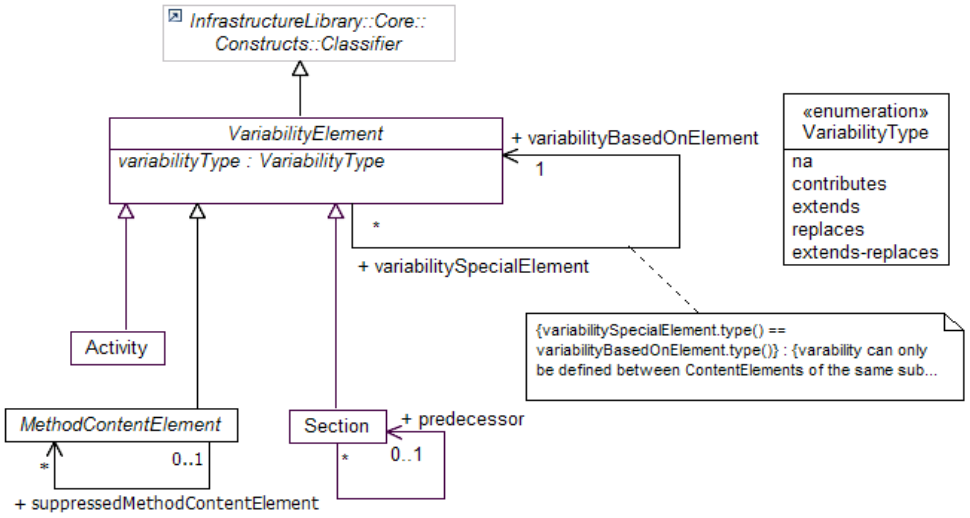


Figure 14.15 - Overview to Variability for Content Elements and Processes

Description

Variability Element is an abstract class derived from Classifier that provides capabilities for content variation and extension to a specific list of SPEM 2.0 classes. It defines a superclass for Activity (Section 9.1), and Section 11.6, and Method Content Element (Section 12.4) in the overall SPEM 2.0 taxonomy of classes using the UML 2 package merge mechanism. The association Variability Specialization shall only be instantiated between two subclasses of Variability Element of the same concrete type. The element on variabilitySpecialElement side of the relationship defines a value for the attribute variabilityType defining the nature of the relationship using a literal from the enumeration Variability Type.

Attributes

- variabilityType: VariabilityType

  If in instance of the variabilitySpecialization association between two Variability Elements of the same type exists, then the variabilityType attribute specifies how the element at the variabilitySpecialElement end of the association changes the Content Element at the variabilityBasedOnElement end. See the Variability Type enumeration class for definitions for the different types of variability.

Association Properties

- variabilityBasedOnElement: VariabilityElement

  This 1-to-many association defines, in combination with the variabilityType attribute, that a number of special Variability Elements of the same type represent a variation of exactly one other Variability Element. See the Variability Type enumeration class for definitions for the different types of variability.

Semantics

Variability Element of the meta-model package Method Plugins adds the capabilities of variation and extension to SPEM Elements that derive from it. Variability provides a mechanism for customizing Variability Elements without actually directly modifying their original structures or textual content. Instead, variability allows you to describe with separate objects the differences (additions, changes, omissions) relative to the original. This Method Plugin concept allows users to factor their method content and processes in interrelated units and even to architect method content and processes in layers that extend each other with new capabilities. The resulting method and process design can be dynamically combined and applied on demand using the interpretation rules defined for Variability Element Specializations. This allows practitioners to assemble the most accurate method and process descriptions possible. It also allows process practitioners to extend and tailor method content and processes they do not own and to easily upgrade to newer versions by simply reapplying their personal changes to these upgrades.

Variability Element defines two types of variability and one type of extension, which are formally defined for the enumeration Variability Type in Section 14.11. Figure 14.16 shows an example of how the first of these types named Contribution can be applied to Core Content Elements, where all Core Content Elements have been derived from the meta-model class Variability Element via package merge.

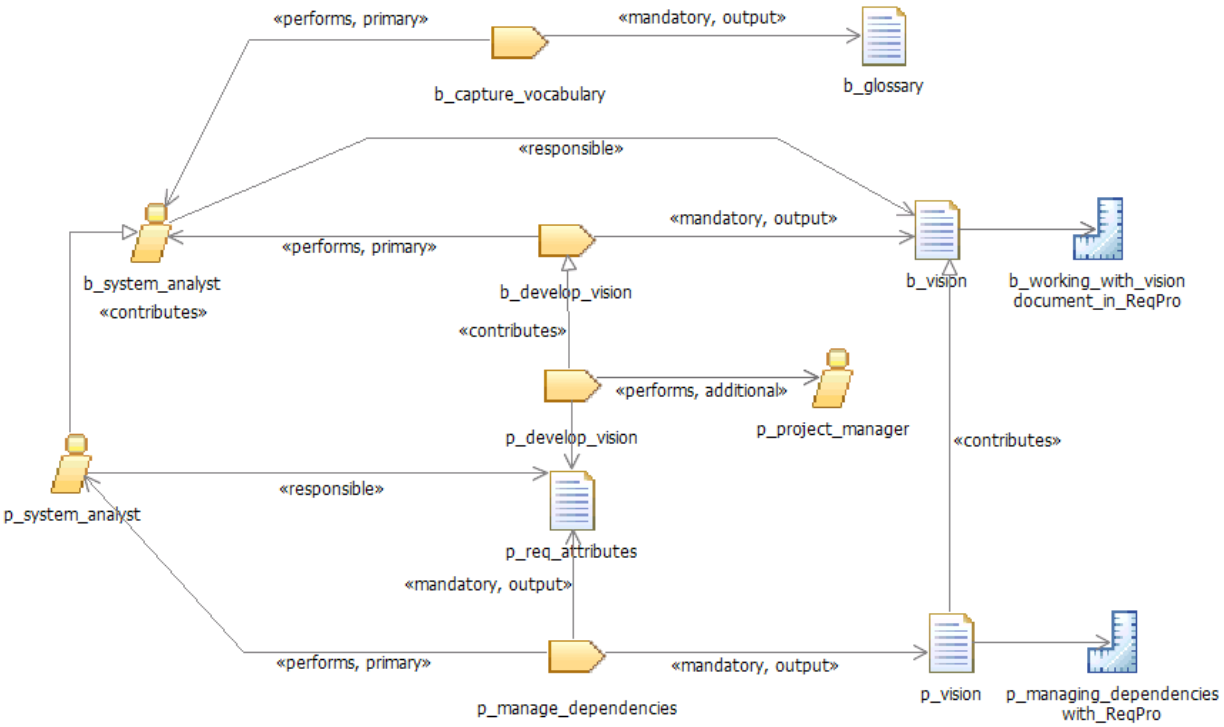


Figure 14.16 - Example for variability using Contribution on Roles, Task, and Work Products

The diagram of Figure 14.16 shows Content Elements from two potential Method Plugins. Elements starting with a "b_" in the name would be placed in a 'Base' Plugin and elements beginning with "p_" would be placed in a Method Plugin that extends the Base-Plugin. The model example depicted covers several cases of using contribution to extend the elements from the base. Overall, it models how the method of requirements management (in particular the part in which a Vision document is being developed) has been extended by the Plugin elements with the new capability of identifying and managing traceability dependencies among requirements. Applying the Method Plugin would add this capability. Removing the Plugin would remove this capability. One can see based in this example that such Method Plugins provide a key way to modularize method content and to dynamically generate specific variants of method content on demand (in this case with or without doing traceability work).

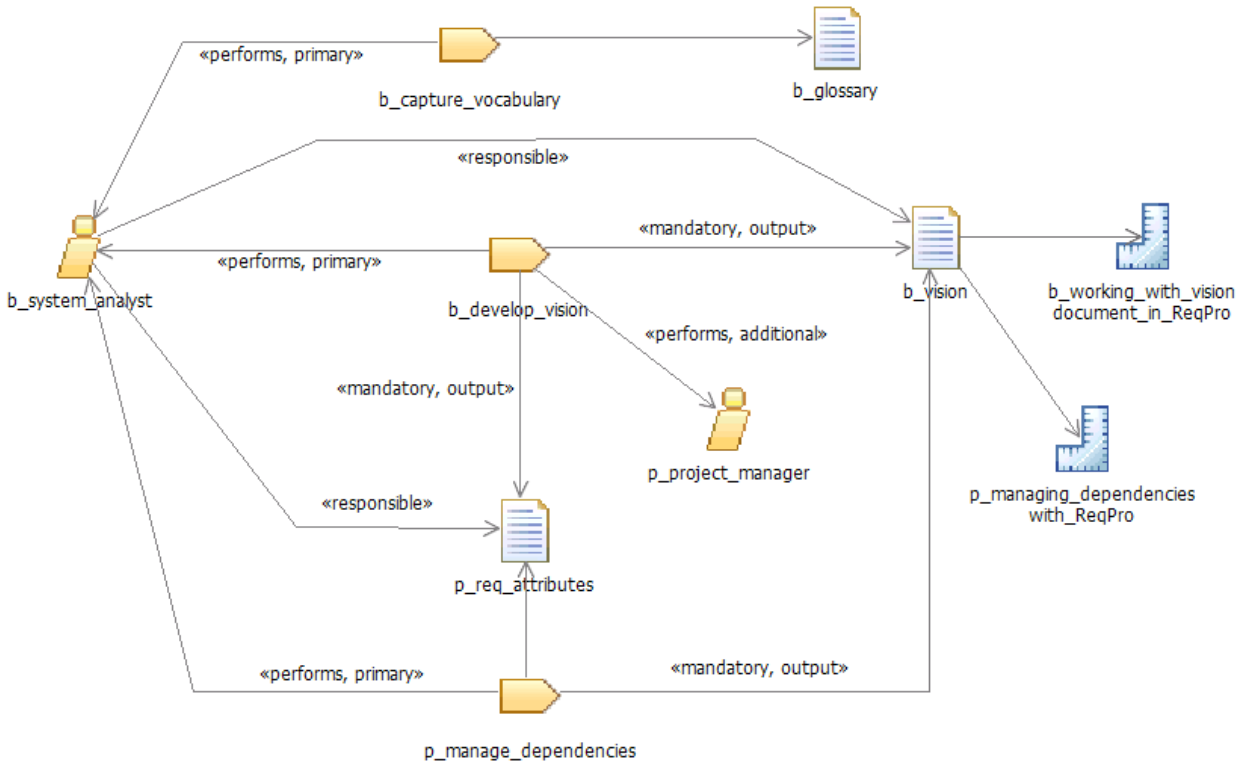


Figure 14.17 - Result after interpreting Contribution in Figure 14-15

Looking at the details of Figure 14.16, one can see that the Task p_develop_vision adds a relationship to a new output Artifact p_req_attributes via the Contributes generalization relationship. The base Task b_develop_vision had only one output Artifact type associated. After interpretation of the 'contributes' relationship, the resulting Task has two Artifact types as output: The vision document and the requirements attributes. Note that in addition to just adding new outputs, a process engineer would also model additional Steps for the Task's textual descriptions to be contributed to the base Task as well. In the same way, the Plugin defines a new Tool Mentor guidance element describing how to use the tool ReqPro to manage traceability dependencies from within the vision document, which is contributed to the base's b_vision artifact. As a result the artifact will have two Tool Mentors associated. Further, it contributes to the b_system_analyst role a new work product responsibility, modeling that the system analyst is also responsible for the new Artifact p_req_attributes.

SPEM 2.0 Profile Notation

This stereotype is abstract and intended to serve as the base for the three concrete stereotypes defined for Variability Type (Section 14.11).

Stereotype	Super-/Metaclass	Keyword	Properties	Abstract	Icon
VariabilitySpecialization	Generalization	n/a	n/a	yes	n/a

14.11 Variability Type

Super Class

n/a : Enumeration

Description

Variability Type is an Enumeration used for values for instances of Variability Element's attribute variabilityType. It defines the nature of how a Variability Element extends another Variability Element.

Enumeration Values

- na — This is the default "not assigned" value of a Variability Element's variabilityType attribute that is set in the case that no variability association is present between the Variability Element and other Variability Elements.
- contributes — Contributes provides a way for instances of Variability Elements to contribute their properties into their base Variability Element without directly altering any of its existing properties, i.e., in an additive fashion. Properties contributed are: attribute values and association instances. The effect after interpretation of contribution is that the base Variability Element is logically replaced with an augmented variant of the element that combines attribute values and association instances. The way this combination is realized depends on the type of the attribute or association. For example, String attributes are concatenated, resolving embedded commands for dependent text or merging text fragments (e.g., descriptions for content elements). Additional elements in to-many associations are added (e.g., additional Guidance elements or Task Uses of an Activity). Different elements in to-one associations are ignored (e.g., the one Primary Performer of a Task). Multiple Variability Elements can contribute to the same base element and all of these contributions properties are added to the base in the same fashion. The following table provides the detailed list of interpretation rules:

attribute values	String values from the special Variability Element are concatenated with values from the based-on Variability Element. Other values from the special Variability Element of any other type such as Integer, Date, URI are ignored. The identifying attributes such as names of Variability Elements are exempt from this rule and will not be modified.
0..1-association instances	A to-one association instance of the based-on Variability Element is replaced with the association instance from the replacing special Variability Element. If the special Variability Element does not have an association instance, then resulting element will also not have an association.
0..n-association instances	Association instances of the special Variability Element are added to the already existing association instances of the based-on element. If both Variability Elements refer to the same object, then only one instance of the association will remain.

- replaces

Replaces provides a way for Variability Elements to define a replacement of a base Variability Element without directly changing any of its existing properties. This is commonly used for Method Plugins that aim to replace specific Content Elements such as Role Definitions, Task Definitions, or Activities with either a complete new variant or to change fundamental relationships of these elements (e.g., Role-Artifact responsibility). Properties replaced are attribute values and association instances. The effect of this is that the base Variability Element is logically replaced with this new variant of the element to which all incoming associations still point as before, but which has potentially new attribute values and outgoing association properties. This provides a very powerful mechanism to replace, for example, whole Activities in a Process with complete new realizations of the Activity. For instance, one can replace an Activity doing use case-based design with an activity doing agile code-centric development, thus doing the same work using a different development technique utilizing different Roles, Tasks, etc. Another example would be to replace an Activity that describes database design for an RDBMS with an Activity that describes database design for an OODBMS. A Variability Element can only be replaced by one other element at a time. For example, if two Method Plugins replace the same element, only one Method Plugin can be used for interpretation (see Method Configuration Concept in Section 14.2, which accomplishes this and for more details on how to resolve such conflicts). The following table provides the detailed list of interpretation rules:

attribute values	Values from the special Variability Element are replaced with values from the based-on Variability Element including unassigned values. This rule also applies to identifying attributes. In other words, after the replacement, the resulting object has the object id of the replacing element.
0..1-association instances	A to-one association instance of the based-on Variability Element is replaced with the association instance from the replacing special Variability Element. If the special Variability Element does not have an association instance, then resulting element will also not have an association. Association instances of the special Variability Element replace all association instances of the based-on Variability Element. If the replacing special Variability Element does not define any association instances, but the base does, then the resulting element will also not have an association instance. The special element could recreate any of the lost association instances if necessary.

- extends

Extension allows Method Plugins to easily reuse elements from a Base Plugin by providing a kind of inheritance for the special Variability Element. Attribute values and Association instances are inherited from the based-on Variability Element to the special Variability Element. The result of interpretation is that the special element has the same properties as the based-on has, but might override the inherited properties with its own values. Hence, extends is not used to modify content of the base Plugin, but to provide the ability for the extending Plugin to define its own content, which is a variant of content already defined (e.g., a special version of a generic Review Record for a specific type of review). The effect of this is that the base Variability Element and any number of extending Variability Elements can be used side by side, but refer to each other via the extends relationship.

attribute values	Values from the based-on Variability element are inherited and used to populate the special Variability Elements attributes. If the special element attributes are already populated, the inherited values are ignored. The identifying attributes are exempt from this rule and will not be modified.
0..1-association instances	The one association instance of the based-on Variability Element is inherited to the special Variability Element. If the special Variability Element defines its own association instance, then the inherited one is ignored.
0..n-association instances	Association instances defined for the based-on Variability Element are inherited to the special Variability Element. If the special Variability Element defines its own association instances, then the inherited ones are ignored.

- extends-replaces

Extends-replaces combines the effects of extends and replace variability into one new variability type. Whereas the replaces variability completely replaces all attributes and outgoing association instances of the base variability element with new values and instances, or removes all values or association instances if the replacing element does not define any. Extends-replaces variability only replaces the values that have been redefined and leaves all other values of the base element as is. In other words, extends-replace allows selectively replacing only parts of the base elements attributes and association instances. This type of variability can be used for method plug-ins that, for example, would like to rename elements or replace some descriptions with new ones without completely remodeling all other relationships and attributes it needs to keep intact.

attribute values	Values from the special Variability Element are replaced with values from the based-on Variability Element not including unassigned values. This rule also applies to identifying attributes. In other words, after the replacements the resulting object has the object id of the extends-replaces element.
0..1-association instances	A to-one association instance of the based-on Variability Element is replaced with the association instance from the replacing special Variability Element. If the special Variability Element does not have an association instance, then resulting element will keep its association instance from the base element.
0..n-association instances	Association instances of the special Variability Element replace all association instances of the based-on Variability Element. If the replacing special Variability Element does not define any association instances, but the base element does, then the resulting element will keep these association instances.

Semantics

SPEM 2.0 supports four different types of variability, which are represented in the UML2 profile as different Generalization stereotypes. Each variability type extends the semantics of UML2 generalization as described below. Varability can only be defined between Variability Elements of the same subtype, e.g., Role Definition with Role Definition, but not Role Definition and Task Definitions.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
VariabilityContributes	VariabilitySpecialization	contributes	n/a	no	n/a
VariabilityReplaces	VariabilitySpecialization	replaces	n/a	no	n/a
VariabilityExtends	VariabilitySpecialization	extends	n/a	no	n/a
VariabilityExendsReplaces	VariabilitySpecialization	extends-replaces	n/a	no	n/a

14.12 Work Product Port

Super Class

Process Element

Description

A Work Product Port defines the work products input and outputs for a Process Component. It is defined based on exactly one type of Work Product and defines for exactly one Process Component if this Work Product type is to be expected as a required (input) or supplied (output) by the Process Component. It also specifies if this input or output is optional or not.

Attributes

- portKind : ParamterDirectionKind = inoutput — This attribute defines if the port represents an input or output Work Product.
- isOptional: Boolean = false — This attribute specifies if the port represents a mandatory or optional Work Product input or output.

Association Properties

- portType: WorkProduct — This association defines the exact type of the Work Product Port.

Semantics

No additional Semantics defined.

SPEM 2.0 Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
WorkProductPort	ProcessElement / Port	work product port	portKind, isOptional	no	n/a

14.13 Work Product Port Connector

Super Class

Process Element

Description

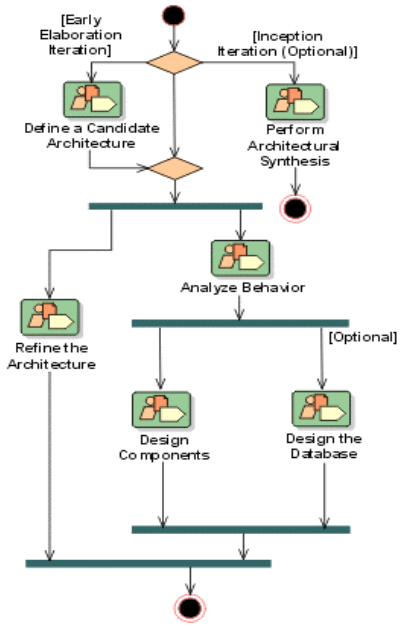
A Work Product Port Connector is used to connect Work Product Ports for assembling Process Components. See Section 14.7 for more details and examples.

Association Properties

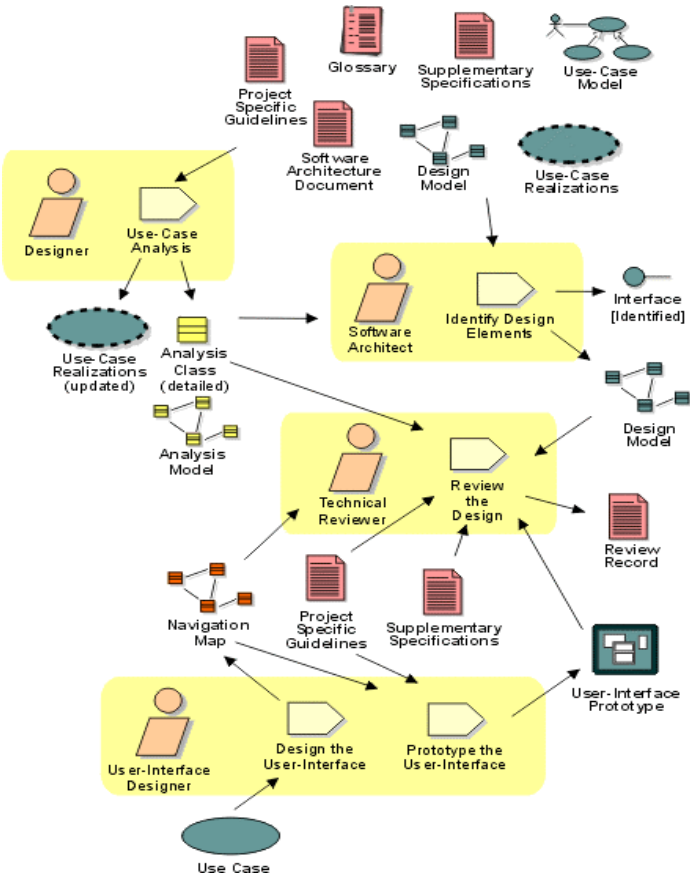
- connectedPort: WorkProductPort — This association connects many Work Product Ports with many other Work Product Ports.

15 Process Diagrams

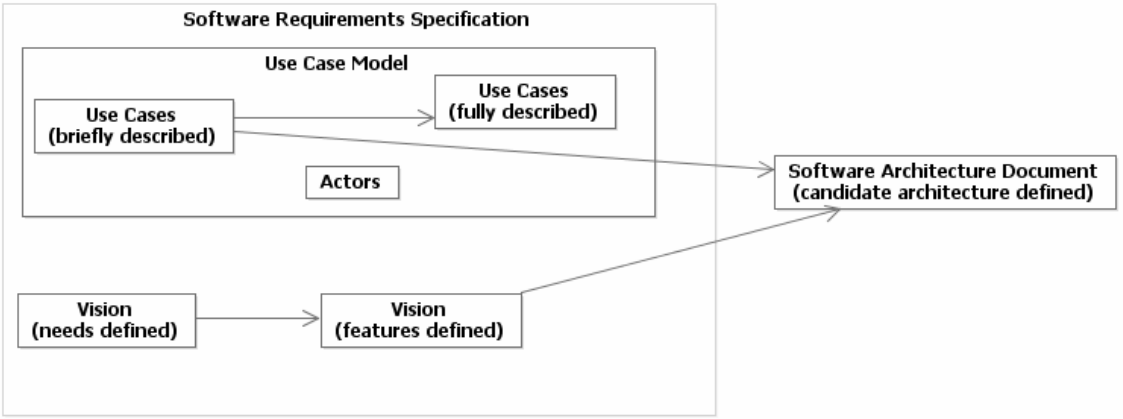
15.1 Workflow Diagram



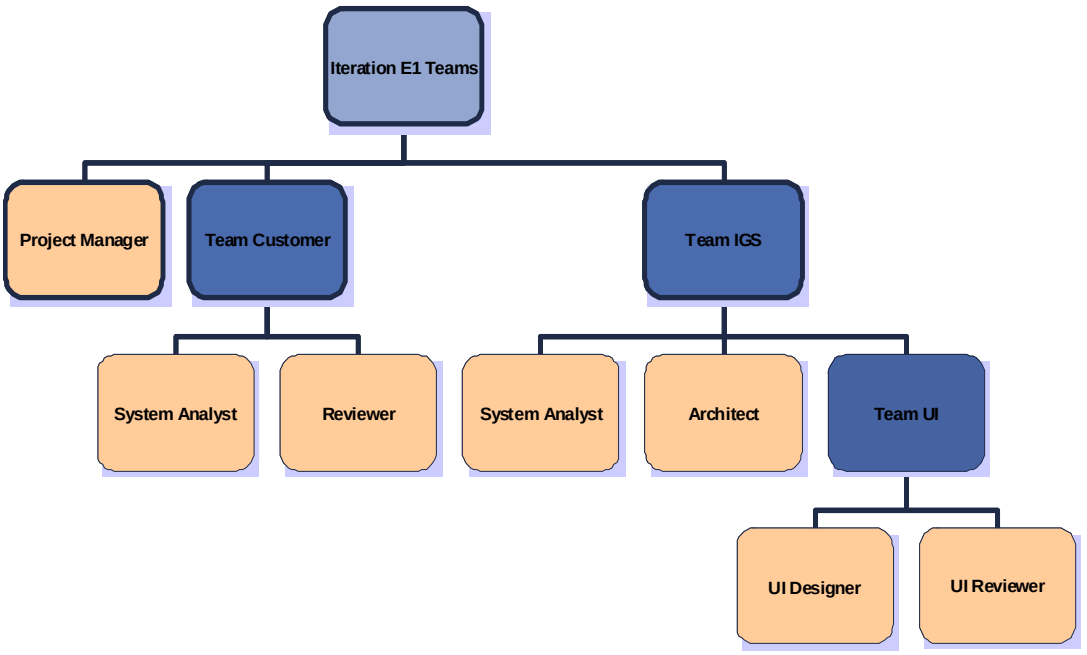
15.2 Activity Detail Diagram



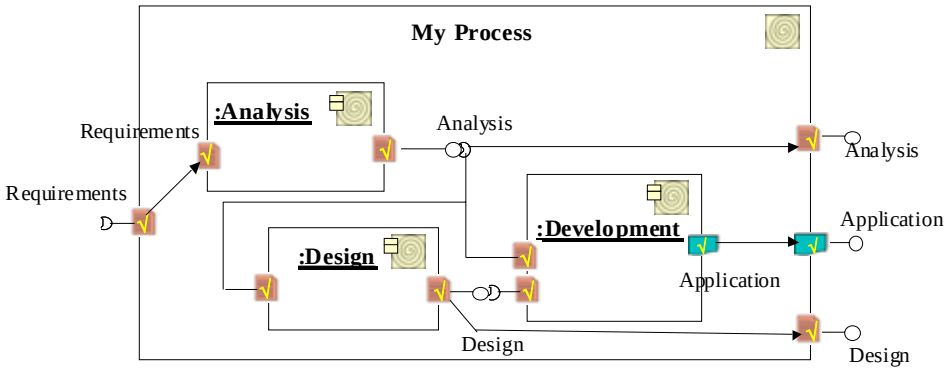
15.3 Work Product Dependency Diagram



15.4 Team Profile Diagram



15.5 Process Component Diagram



16 Enacting SPEM 2.0 Processes

This chapter summarizes SPEM 2.0 capabilities and mechanisms for process enactment and illustrates different scenarios for enacting processes created with the SPEM 2.0 meta-model.

Process described with the SPEM 2.0 meta-model can be enacted in different ways. The two most common ways of enactment are:

- Mapping the processes into Project Plans and enacting these with project planning and enactment systems such as IBM Rational Portfolio Manager or Microsoft Project (Section 16.1).
- Mapping the process to a business flow or execution language and then executing this representation of the processes using a workflow flow engine such as a BPEL-based workflow engine (Section 16.2).

16.1 Process Enactment with Project Planning Systems

The following examples illustrate how SPEM 2.0 supports process enactment with planning and resource management tools such as IBM Rational Portfolio Manager or Microsoft Project by providing the means to map Processes to project plans. A SPEM 2.0 Process can be systematically mapped to a project plan by instantiating the different Process' breakdown structure views.

As illustrated in Figure 16.1, a Process can be represented as an Activity diagram (see Section 7.2) as well as a Breakdown Structure (see Section 9.1). A work breakdown structure can be mapped directly to a plan using a mechanism called 'instantiation.' Deriving a plan from a process is an interactive procedure in which a project planner substitutes concrete work product instances for the process' work product type declarations as well as resources for roles. If work product types have multiple instances, then certain Tasks defined in the process need be replicated to allow assignment of different resources working on these different work products (e.g., the Task 'Use Case Analysis' in Figure 16.1 is replicated for every use case defined for the concrete project or every resource assigned to work on use cases).

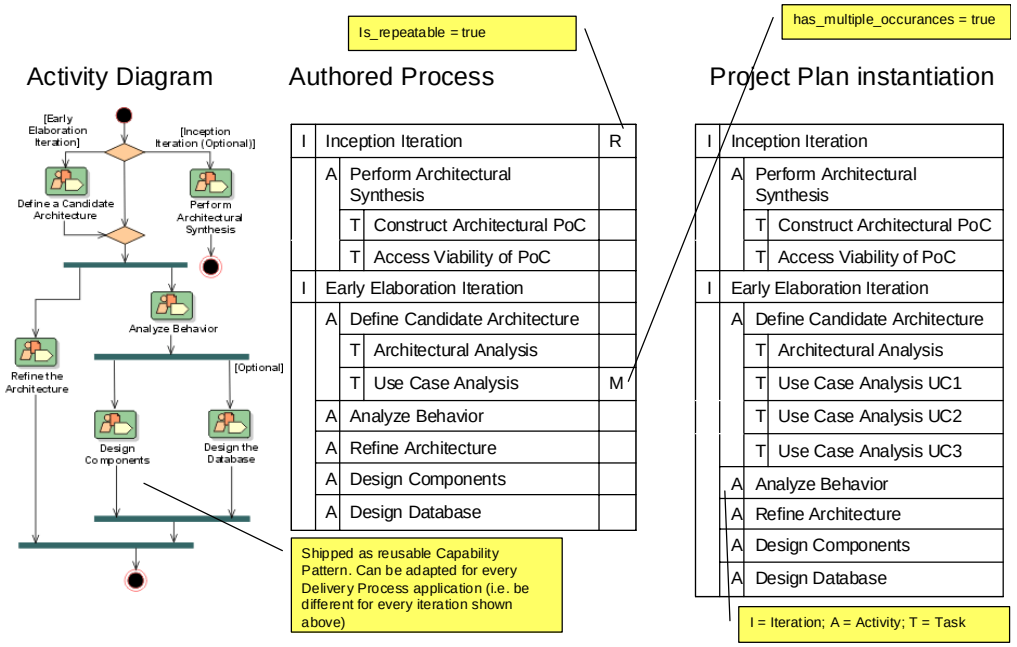


Figure 16.1 - Mapping between Activity Diagram, Process, and Project Plan

Processes defined using SPEM 2.0 breakdown structures provide key information attributes that provide the Project Planner with the right guidance to make these instantiation decisions (see attributes defined for Breakdown Elements, Section 13.1 and Work Breakdown Element, Section 13.3). For example, a hasMultipleOccurrence attribute indicates that a Breakdown Element such as a Task Use will be mapped to multiple plan tasks (or summary tasks) as in the example of Figure 16.1. An isRepeatable flag indicates for a Work Breakdown Element, such as an Activity, that it defines an Iteration and that these Iterations will be repeated one after the other (in contrast to the multiple occurrences elements that could be performed in parallel by different resources). The example in Figure 16.2 shows several examples of repeatable Work Breakdown Elements such as the ‘Build Cycle’ or the nested ‘Program and Test Cycle.’

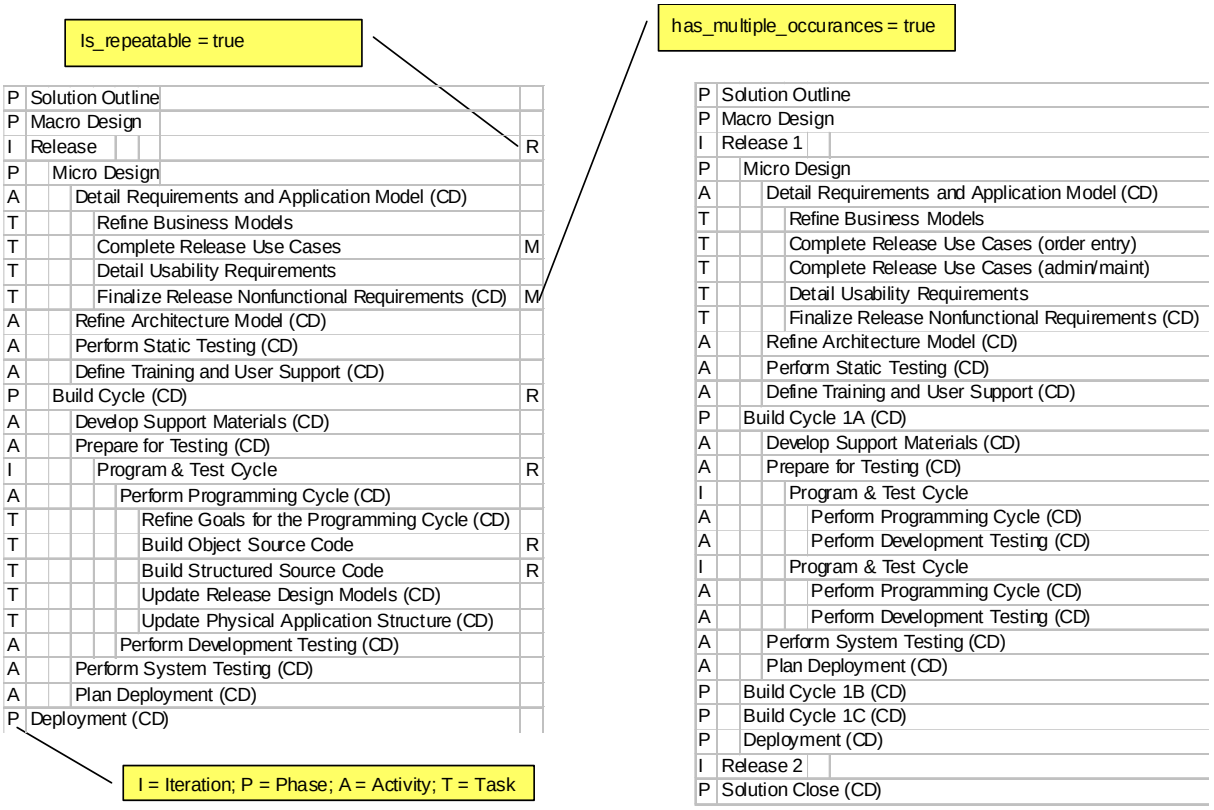


Figure 16.2 - A second Process to Plan mapping example

In addition to these work-centric plans, an alternative mapping could take the work product breakdown structure and map it to plan by mapping Activities to a plan's summary tasks and Work Products to the plan's tasks. Resource profiles can then be allocated to the plan's task by using the responsible and modifies relationship defined between Work Products and Roles. Plan instantiation would then map the Work Products to concrete work product instances, the work product entry and exist states to task annotations, and Roles to concrete resources. The resulting plans would be suitable for a very agile project environments that do not intend to follow strict plans of activities that prescribe the work to be done step-by-step, but rather just define which resource produces which work products (e.g., features or stories) in which state for which date.

16.2 Process Enactment with a Workflow Engine

The separation of SPEM 2.0 models from behavior models (see Chapter 10) opens up the possibility to utilize enactment machines for many different kinds of behavior modeling approaches. In other words, as SPEM 2.0 models can be linked to behavior model elements from any behavior modeling approach, any enactment machine created for these behavior models can be used.

For example, activities in a SPEM 2.0 delivery process could be linked to one or more BPMN diagrams. An enactment machine for BPMN models can then be used to run these models.

17 Migrating SPEM 1.1 Models to SPEM 2.0

SPEM version 2.0 is a major revision of SPEM version 1.1. Many issues and defects have been fixed in version 2, new capabilities have been added, as well as gaps and limitations of version 1.1 have been closed. SPEM 1.1 was based on UML 1.4 and SPEM 2.0 is now based on UML version 2.0, which represents a major revision of the UML.

The following table provides an overview of how the concrete SPEM 1.1 meta-classes defined in this specification map to SPEM 2.0 classes. It lists the SPEM 1.1 concepts in the order as they have been defined in the SPEM 1.1 specification [SPEM 1.1]. It shows the corresponding SPEM 2.0 concepts as well as provides brief explanations and rationale of the changes. If specific RFP (OMG ad/2004-11-04) requirements have been addressed, then it lists the requirement's section number in parentheses. For example: "(RFP 1.2.3)" would indicate that the requirement or constraint described in "Section 1.2.3" of the RFP has been addressed or that the change relates to it.

SPEM 1.1	**SPEM 2.0**	**Comment / Change Rationale**
External Description	Content Description	Renamed, because the meaning of 'external' was not clear and implied a design decision. Removed 'language' attribute as localization can be implemented in different ways independent of such an attribute. Removed 'medium' as implementations will have their own design dependent realizations of this information.
Guidance	Guidance	Same in both specifications.
Guidance Kind	Kind (instance)	The specific Guidance Kind has been replaced with a generic Kind mechanism that can be applied to any SPEM element. The Base Plugin defined in Chapter 18 provides predefined stereotypes for Guidance Kinds for backwards compatibility.
Categorizes Dependency	Category	Changed from 'package to model element dependency' to Category classifier that manages associations to all categorized elements. The SPEM 1.1 dependency did not work well with packages, because process modelers wanted to define multiple categories that all categorize the same element independent of package boundaries and strict compositions.
Impacts Dependency	Work Product Relationship	Replaced with a generic Work Product Relationship. The Base Plugin defined in Chapter 18 provides an 'impacted by' stereotype for backwards compatibility.
Import Dependency	<UML 2 Infrastructure> Element Import	Reused from UML 2 Infrastructure. (RFP 6.6.8)
Precedes Dependency	Work Sequence	Made a bidirectional association for improved navigability. Added missing fourth type (start-finish).
Refers To Dependency	<deprecated>	This dependency required modeling redundant information in the process model which leads to maintenance problems. Deprecated for lack of usefulness (RPF 6.6.12).

Trace Dependency	<deprecated>	Deprecated because of lack of semantics and overlap with other concepts (such as Impacts). Specific traces have been introduced such as the trace for Method Content Use to Method Content Definitions.
Work Product	Work Product	Almost unchanged. Deliverable flag did not work for many users because deliverables normally package parts of many other work products with additional content describing the delivery. SPEM 2.0 defined general Work Product Relationship as well as provides a Deliverable stereotype as part of the Base Plugin (Chapter 18).
Work Product Kind	Kind (instance)	The specific Work Product Kind has been replaced with a generic Kind mechanism that can be applied to any SPEM element. The Base Plugin defined in Chapter 18 provides predefined stereotypes for Work Product Kinds for backwards compatibility.
Work Definition	Work Definition / Activity	Derived from Classifier, not Operation anymore to utilize the new UML 2 ability to model behavior as standalone classifier. This adds more flexibility to the way work is represented in SPEM (e.g. with multiple performing roles) (RFP 6.6.7, RPF 6.6.12). Made it abstract class to avoid the confusion of SPEM 1.1 of when to use Activity and when to use Work Definition (RPF 6.6.12). Introduced more subclasses to support the differentiation of work definitions representing method content versus work definitions in processes.
Activity Parameter	Work Definition Parameter	Renamed, because every work definition can have parameters not only activities.
Activity	Activity / Task Definition / Task Use	For implementer of the process structure meta-model package (e.g. for compliance point “Process with Behavior and Content”) no real change to SPEM 1.1 applies on the activity concept except that there is no separate notion of Steps anymore. Steps would be expressed as sub-activities assuming that a process is only described by an n-level breakdown of just activities providing a simpler view on process to the end-user than SPEM 1.1. For implementers of the “Complete” compliance point, SPEM 2.0 adds the separation of method content definitions from the use of method content in processes to SPEM 1.1. Method content is defined with task definitions and can be applied in many different processes composed of activities that refer to the method content via task usages. For that purpose the SPEM 1.1 concept of activity has been refined into activity, task definition, and task use. The word task was used to apply a terminology familiar to many project-management oriented users of SPEM as well as BPMN as the non-decomposable unit of work.

Step	Step	Same as SPEM 1.1, but now derived from Work Definition to represent them in UML 2 Activity diagrams with actions and activities, instead of action states as in the UML 1.4-based SPEM 1.1, which are deprecated in UML 2 (RFP 6.2.1).
Process Performer	Role Use (Composite Role, Team Profile)	In SPEM 1.1 Process Performer was a generalization of Process Role that would physically own work definitions as operations, in particular high-level work definitions such as Iterations. Process Performer led to the creating of many special instances with low reusability (RPF 6.6.12). SPEM 2 does not separate the Process Performer and Process Role anymore, but allows assigning Role Use instances to any Work Definition reducing the number of objects defined for processes as well as increasing reusability of these concepts. For modeling more general roles or role groupings SPEM 2.0 introduces the concepts of Composite Role and Team Profile instead. The concepts provide more flexibility in terms of defining and reusing them independent of specific Work Definitions. Process Performer required strict compositions.
Process Role	Role Definition / Role Use	For implementers of the compliance point "Process with Behavior and Content," the Role Use concept works equivalently to the Process Role concept of SPEM 1.1. When implementing the "Complete" compliance point, SPEM 2.0 distinguishes the reusable definition of a role from its concrete use in a process increases reusability and adaptability at the same time. A Role is then defined and fully documented in one place. It can then be referred to via Role Usages in many different places of many different processes. Each Role Use can define its own context specific properties. For an example, a Designer role might be responsible in one particular process or one phase of a process for an Analysis and a Design Model, and in another process or another phase of the same process responsible for a Design Model only, because that process or phase does not require the creation and maintenance of an Analysis Model. SPEM 1.1 did not provide the flexibility of reusing and adapting roles, but required the definition of separate new roles each time new properties were required.
Package	Method Package / Process Package	Derived from the UML 2 package with additional constraints that enforce the physical separation of method content and process definitions.
Process Component	Process Component	Removed flawed self-containedness constraints and "Unification" mechanism of SPEM 1.1 and replaced with Ports and Work Product correspondence concepts as part of the component definition. (RPF 6.6.12, RFP 6.7.1)

Process	Process	In SPEM 2 Process is not defined as a specialization of Package as in SPEM 1.1, but as a Work Definition to be consistent with other OMG definitions of process (e.g., BPMN, BPDM drafts). A process is generally seen in these specifications as the model or structure of objects that describe a behavior rather than just a physical container for these behavioral models.
Discipline	Category (instance)	Defining discipline as a specialization of Package in SPEM 1.1 created unnecessary restrictions on the way elements could be packaged. In SPEM 2, Disciplines can be represented as one instance of Category that allows any element in any package to be categorized to be part of a specific Discipline independent of its physical location. The Base Plugin provides a stereotype for Discipline for backward compatibility.
Phase	Kind (instance)	Phases can be represented in SPEM 2.0 as normal Activities. One can use the general Kind mechanism to qualify such Activities as Phases. The Base Plugin provides a stereotype for Phase for backward compatibility.
Lifecycle	Process	In SPEM 2 Process is not defined as a Package anymore, but as a specialization of Work Definition. The SPEM 1.1 Lifecycle concept would have been synonym to the SPEM 2 Process concept and was therefore redundant and deprecated.
Iteration	Kind (instance)	Iterations can be represented in SPEM 2.0 as normal Activities. One can use the general Kind mechanism to qualify such Activities as Phases. The Base Plugin provides a stereotype for Phase for backward compatibility.
Precondition	Precondition	No change except for reusing UML 2 constraints (RFP 6.2.1). The specializations of Constraint in SPEM 1.1 were unnecessary because they did not provide any additional properties and have been removed. SPEM 2 defines direct associations to the UML 2 Constraint class.
Goal	Postcondition / Purpose	The term Goal caused confusion. Postcondition is less ambiguous. Purpose is a text attribute of all content elements in SPEM 2, expressing the purpose of a process element within its specific context.

18 The SPEM 2.0 Base Plug-in

The SPEM 2.0 Base Plug-in is a pre-defined Method Plugin (created as an instance of Method Plugin defining instances of SPEM 2.0 meta-model classes) that already provides commonly used instances for many SPEM 2.0 meta-model concepts for the domain of Software Engineering. It is intended to provide a starting point and provide potential reusable assets for software development processes. SPEM 2.0 provides, in addition to this Method Plug-in, a profile that contains stereotypes for many of the elements defined here, especially for reusable Kind classes that are meant to express stereotypes for use-define process models.

18.1 Activity Kinds

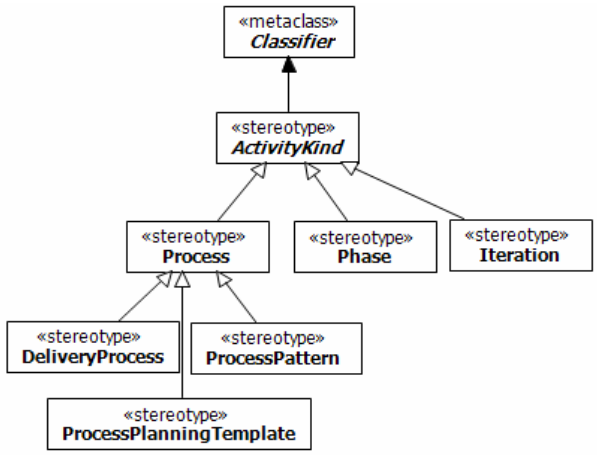


Figure 18.1 - The Activity Kinds Stereotypes

Activity Kinds provides the capability for a process engineer to define life-cycle models using the terminology they are used to. For example, if a process engineer would like to distinguish specific levels of the breakdown to represent a special kind of Activities, s/he can define instances Activity Kinds and assign these to the Activities. ‘Phase’ and ‘Iteration’ are popular examples for Breakdown Element Kinds. They would be represented in a breakdown structure as Activities with respective Kinds assigned. Another example can be found in the IBM Rational’s Summit Ascendant Method. The first level of Summit’s breakdown structure is not referred to as Activity, but as a ‘Module.’ The Kind class allows defining and modeling such specific interpretation of the breakdown level.

Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
ActivityKind	Classifier	n/a	n/a	yes	n/a

18.1.1 Phase

Phase represents a significant period in a project, ending with major management checkpoint, milestone, or set of Deliverables. It is included in the model as a predefined special Activity, because of its significance in defining breakdowns. Typically, Phase is an Activity for which its attribute isRepeatable attribute is set to ‘False.’

Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
Phase	ActivityKind	phase	n/a	no	SPEM 1.1 backwards compatibility icon:

18.1.2 Iteration

Iteration groups a set of nested Activities that are repeated more than once. It represents an important structuring element to organize work in repetitive cycles. The concept of Iteration can be associated with different rules in different methods. For example, the IBM Rational Unified Process method framework (RUP) defines a rule that Iterations are not allowed to span across Phases. In contrast, IBM Global Services Method (GSMethod), based method frameworks, this rule does not apply and Iterations can be defined that nest Phases. (Note: Any Breakdown Element can be repeated; however, Iterations play a special role for repeated units of work with key milestones.) Typically, Iteration is an Activity for which the default value for its attribute isRepeatable is ‘True.’

Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
Iteration	ActivityKind	iteration	n/a	no	

18.1.3 Process

A Process is a special Activity that describes a structure for particular types of development projects or parts of them. To perform such a development project, a Process would be adapted for the specific organizational or project situation and then ‘instantiated’ by assigning concrete resources to Role Uses, creating multiple instances for Work Product Uses, etc.

Development processes define how development projects shall be executed. Development processes are recognized as sequences of phases and milestones, expressing a lifecycle of the product under development. Processes also define how to get from one milestone to the next by defining sequences of work, operations, or events that usually take up time,

expertise, or other resources and which produce some outcome. Processes are represented in SPEM 2.0 as a set of partially ordered work definitions intended to reach development milestones, such as the release of a specific software system. A process focuses on the lifecycle and the sequencing of work in breakdown structures.

Reference frameworks for development processes such as CMMI define different levels of maturity for processes. Each level entails different characteristics for the process definition as well as enactment in an organization or project for each level. For example, CMMI defines a “managed process” as performed activities that can be recognized as implementations of development practices. Such a process has certain characteristics: it is planned and executed in accordance with policy; it employs skilled people having adequate resources to produce controlled outputs; it involves relevant stakeholders; it is monitored, controlled, and reviewed; and it is evaluated for adherence to its process description. By contrast, a “defined process” is a managed process that is tailored from the organization’s set of standard processes according to the organization’s tailoring guidelines. A defined process has a maintained process description and contributes work products, measures, and other process-improvement information to the organizational process assets. SPEM 2.0 processes aim to support these characteristics with its reuse (Section 9.2), customization (Section 8.1, Section 8.2), descriptive specification (Section 9.3, Section 9.10), and variability (Section 14.6) features for tailoring and dynamic assembly of processes based on reusable process patterns of best practices.

Profile Notation

Stereotype	Superclass	Keyword	Properties	Abstract	Icon
Process	ActivityKind	process	n/a	no	

18.1.4 Delivery Process

Description

A Delivery Process is a special Process describing a complete and integrated approach for performing a specific project type. It describes a complete project lifecycle end-to-end and shall be used as a reference for running projects with similar characteristics as defined for the process.

A Delivery Process is a Process that covers a whole development lifecycle from beginning to end. A Delivery Process shall be used as a template for planning and running a project. It provides a complete lifecycle model with predefined phases, iterations, and activities that have been detailed by sequencing referencing method content in breakdown structures. It is defined on the basis of experience with past projects or engagements, and/or the best practice use of a development or delivery approach. It defines what gets produced, how those items are produced, and the required staffing in the form of integrated Work, Work Product, and Team Breakdown Structures. For example, a process engineer can define alternative Delivery Processes for software development projects that differ in the scale of the engagement and staffing necessary, the type of the software application to be developed, the development methods and technologies to be used, etc.

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
DeliveryProcess	/ Process	delivery process	n/a	no	

18.1.5 Process Pattern

A Process Pattern is a special Process that describes a reusable cluster of Activities in a general process area that provides a consistent development approach to common problems. Examples of Process Patterns include “use case-based requirements management,” “develop components,” “validate build,” or “ongoing management and support.” Typically but not necessarily, Process Patterns have the scope of one discipline providing a breakdown of reusable complex Activities, relationships to the Roles that perform Tasks within these Activities, as well as to the Work Products that are used and produced. A Process Pattern does not relate to any specific phase or iteration of a development lifecycle, and should not imply any. In other words, a pattern should be designed in a way that it is applicable anywhere in a Delivery Process. This enables its Activities to be flexibly assigned to whatever phases there are in the Delivery Process to which it is being applied.

It is a good practice to design a Process Pattern to produce one or more work product use representing a deliverable. This enables the process engineer to select and integrate Patterns by deciding which deliverables are required as outputs and inputs.

Key applications areas of / areas of reuse for Process Patterns are:

- To serve as building blocks for assembling Delivery Processes or larger Process Patterns. Normally developing a Delivery Process is not done from scratch but by systematically applying and binding patterns. In addition to the standard pattern application of ‘copy-and-modify,’ which allows the process engineer to individually customize the pattern’s content to the particular situation it is applied for, Process Variability concepts (Section 9.2) introduce even more sophisticated inheritance relationships that support dynamic binding of patterns (i.e., the pattern is referenced and not copied). This unique new way of reusing process knowledge allows to factor out commonly reoccurring Activities into patterns and to apply them over and over again in a process. When the pattern is being revised or updated, all changes automatically will be reflected in all pattern application in all processes because of the dynamic binding.
- To support direct execution in a development project that does not work following a well-defined process, but works based on loosely connected process fragments of best practices in a flexible manner (e.g., Agile Development).
- To support process education by describing knowledge for a key area such as best practices on how to perform the work for a Discipline (e.g., Requirements Management), for a specific development technique (aspect-oriented development), or a specific technical area (e.g., relational database design) that is used for education and teaching (e.g., see Figure 18.2 providing an example pattern for the Requirements Management Discipline).

Figure 18.2 shows an example Process Pattern from a work breakdown perspective representing the Requirements Management discipline of the IBM Rational Unified Process. It provides the most typical breakdown structure of doing this work under generic or normal circumstances (including Role and Work Product Uses, which are not presented here). Different applications of this pattern in different Delivery Processes might define individual differences. Also when such a pattern is applied more then once, e.g., in every phase and iteration of an iterative development process, every application might focus on different parts of this pattern and leave out different parts of the pattern in every pattern application.

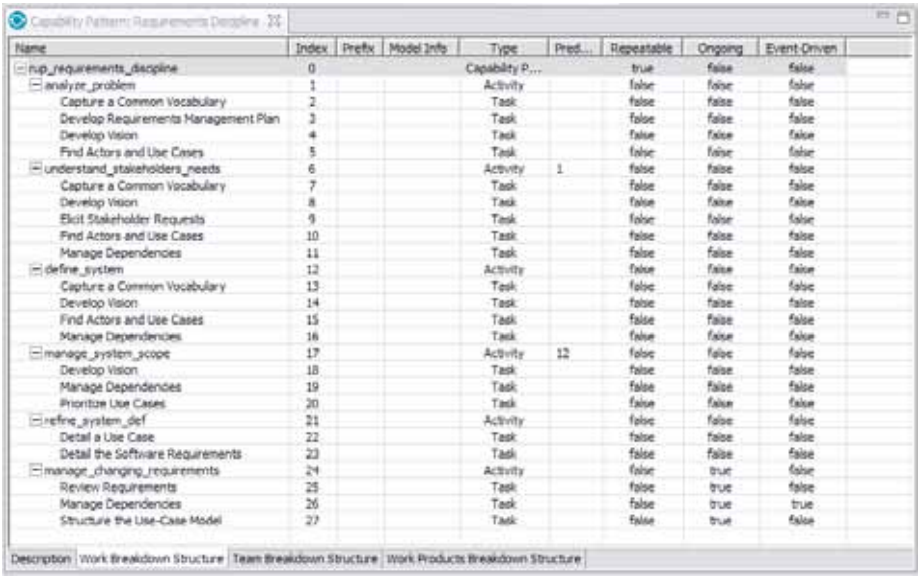


Figure 18.2 - Example for Process Pattern defining Requirements Management work

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
ProcessPattern	/ Process	process pattern	n/a	no	

Figure 18.3 shows an example using the SPEM 2.0 Profile Notation representing a Delivery Process named RUP that has been assembled using the four Process Patterns depicted in the lower right-hand corner. We see that a Capability itself can be assembled using patterns. In the diagram a pattern describing relational database (RDBMS) design has been applied to the OOAD pattern. Further, the diagram shows that two Process Planning Templates (Section 18.4.3) for different types of Iteration plans have been derived from the RUP Delivery Process.

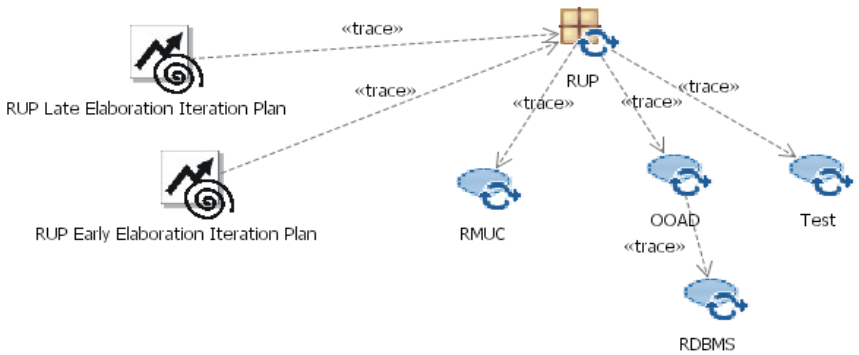


Figure 18.3 - Relationships among Processes

18.1.6 Process Planning Template

A Process Planning Template is a special Process that is prepared for instantiation by a project planning tool. Typically, it is created based on a Process such as a Delivery Process as a whole (e.g., in case of a waterfall-based development approach) or in parts (e.g., in case of an iterative development approach).

A Process Planning Template represents a partially finished plan for a concrete project. It uses the same information structures as all other Process Types to represent templates for project plans. However, certain planning decisions have already been applied to the template, and information has been removed and/or reformatted to be ready for export to a specific planning tool. Examples for such decisions are: a template has been created to represent a plan for a particular Iteration in an iterative development project as depicted in Figure 18.3, which distinguishes early from late iterations in the Elaboration phase of a project; if the targeted planning tool cannot represent input and output of Task, then these have been removed from the structure; certain repetitions already have been applied, e.g., stating that a cycle of specific Task grouped in an Activity have to be repeated n-times; etc.

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
ProcessPlanningTemplate	/ Process	process planning template	n/a	no	

18.2 Category Kinds

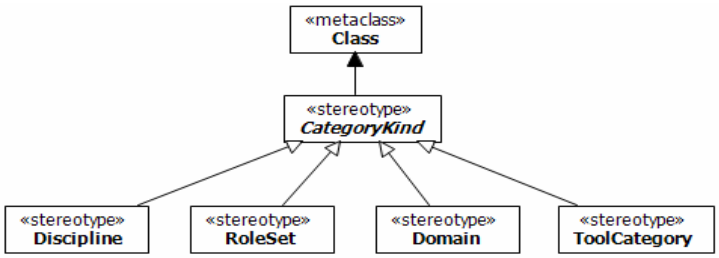


Figure 18.4 - The Category Kinds Stereotypes

Category Kinds are a flexible way of defining different groupings for Content Categories.

Stereotype	Meta-/ Superclass	Keyword	Properties	Abstract	Icon
CategoryKind	Class	n/a	n/a	yes	n/a
Discipline	CategoryKind	discipline	n/a	no	n/a
RoleSet	CategoryKind	role set	n/a	no	n/a
Domain	CategoryKind	domain	n/a	no	n/a
Tool Category	CategoryKind	tool category	n/a	no	n/a

18.2.1 Discipline

A Discipline is a categorization of work (i.e., Tasks for Method Content), based upon similarity of concerns and cooperation of work effort.

A discipline is a collection of Tasks that are related to a major ‘area of concern’ within the overall project. The grouping of Tasks into disciplines is mainly an aid to understanding the project from a ‘traditional’ waterfall perspective. For example, it is more common to perform certain requirements activities in close coordination with analysis and design activities. Separating these activities into separate disciplines makes the activities easier to comprehend.

18.2.2 Role Set

A Role Set organizes Roles into categories.

Role Set is used to group roles together that have certain commonalities. For example, the “Analysts” Role Set could group the “Business Process Analyst,” “System Analyst,” as well as “Requirements Specifier” roles. All of these work with similar techniques and have overlapping skills, but are required as distinct roles for a method (e.g., the method the IBM Rational Unified Process is based on).

18.2.3 Domain

Domain is a refineable hierarchy grouping related work products. In other words, Domains can be further divided into sub-domains, with work product elements to be categorized only at the leaf-level of this hierarchy.

Domain is a logical grouping of work products that have an affinity to each other based on resources, timing, or relationship. A Domain may be divided into subdomains. For example, GS Method uses six predefined Domains for Work Products: Application, Architecture, Business, Engagement, Operations, and Organization.

18.2.4 Tool Category

A Tool Category is a container/aggregate for Tool Mentors. It can also provide general descriptions of the tool and its general capabilities.

18.3 Guidance Kinds

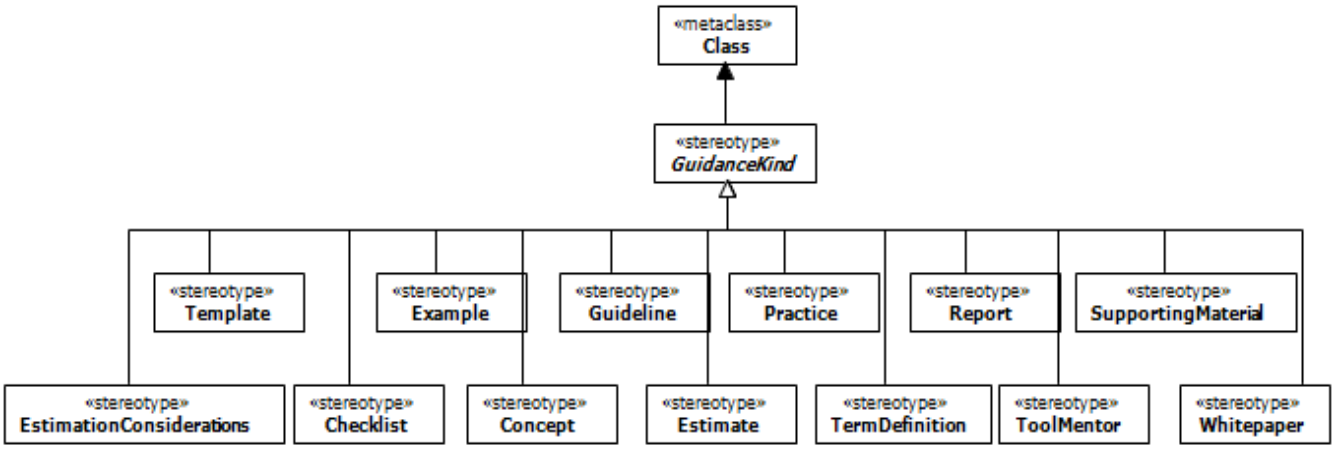


Figure 18.5 - The Guidance Kinds Stereotypes

The following section provides the semantics for commonly used guidance kinds.

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
GuidanceKind	Class	n/a	n/a	yes	n/a
Checklist	GuidanceKind	checklist	n/a	no	n/a
Concept	GuidanceKind	concept	n/a	no	n/a
Estimate	GuidanceKind	estimate	n/a	no	n/a
EstimationConsideration	GuidanceKind	estimation consideration	n/a	no	n/a
EstimationMetric	GuidanceKind	estimation metric	n/a	no	n/a
Example	GuidanceKind	example	n/a	no	n/a
Guideline	GuidanceKind	guideline	n/a	no	n/a
Practice	GuidanceKind	practice	n/a	no	n/a
Report	GuidanceKind	report	n/a	no	n/a
ReusableAsset	GuidanceKind	reusable asset	n/a	no	n/a
Roadmap	GuidanceKind	roadmap	n/a	no	n/a
SupportingMaterial	GuidanceKind	supporting material	n/a	no	n/a
Template	GuidanceKind	template	n/a	no	n/a
TermDefinition	GuidanceKind	term definition	n/a	no	n/a

ToolMentor	GuidanceKind	tool mentor	n/a	no	n/a
WhitePaper	GuidanceKind	white paper	n/a	no	n/a

18.3.1 Checklist

A Checklist is a specific type of guidance that identifies a series of items that need to be completed or verified. Checklists are often used in reviews such as walkthroughs or inspections.

18.3.2 Concept

A Concept is a specific type of guidance that outlines key ideas associated with basic principles underlying the referenced item. Concepts normally address more general topics than Guidelines and span across several work product and/or tasks/ activities.

18.3.3 Estimate (metric kind)

An Estimate is a specific type of Guidance that provides sizing measures, or standards for sizing the work effort associated with performing a particular piece of work and instructions for their successful use. It may be comprised of estimation considerations and estimation metrics.

18.3.4 Estimation Considerations (metric kind)

Estimation Considerations qualify the usage and application of estimation metrics in the development of an actual estimate.

18.3.5 Estimating Metric (metric kind)

Estimation Metric describes a metric or measure that is associated with an element and which is used to calculate the size of the work effort as well as a range of potential labor.

18.3.6 Example

An Example is a specific type of Guidance that represents a typical, partially completed, sample instance of one or more work products or scenario-like description of how Task may be performed. Examples can be related to Work Products as well as Tasks that produce them and any other Content Element.

18.3.7 Guideline

A Guideline is a specific type of guidance that provides additional detail on how to perform a particular task or grouping of tasks (e.g., grouped together as activities), or that provides additional detail, rules, and recommendations on work products and their properties. Among others, it can include details about best practices and different approaches for doing work, how to use particular types of work products, information on different subtypes and variants of the work product and how they evolve throughout a lifecycle, discussions on skills the performing roles should acquire or improve upon, measurements for progress and maturity, etc.

18.3.8 Practice

A Practice represents a proven way or strategy of doing work to achieve a goal that has a positive impact on work product or process quality. Practices are defined orthogonal to methods and processes. They could summarize aspects that impact many different parts of a method or specific processes. Examples for practices would be “Manage Risks,” “Continuously verify quality,” “Architecture-centric and component-based development,” etc.

18.3.9 Report

A Report is a predefined template of a result that is generated on the basis of other work products as an output from some form of tool automation. An example for a report would be a use case model survey, which is generated by extracting diagram information from a graphical model and textual information from documents, and then combines these two types of information into a report.

18.3.10 Reusable Asset

A Reusable Asset provides a solution to a problem for a given context. The asset may have a variability point, which is a location in the asset that may have a value provided or customized by the asset consumer. The asset has rules for usage that are the instructions describing how the asset should be used.

18.3.11 Roadmap

A Roadmap is a special Guidance Kind that is only related to Activities. A Roadmap represents a linear walkthrough of an Activity, typically a Process. An instance of a Roadmap represents important documentation for the Activity or Process it is related to. Often a complex Activity, such as a Process, can be much easier understood by providing a walkthrough with a linear thread of a typical instantiation of this Activity. In addition to making the process practitioner understand how work in the process is being performed, a Roadmap provides additional information about how Activities and Tasks relate to each other over time. Roadmaps are also used to show how specific aspects are distributed over a whole process, providing a kind of filter on the process for this information.

18.3.12 Supporting Material

Supporting Materials is a catch-all for other types of guidance not specifically defined elsewhere. It can be related to all kinds of Content Elements, i.e., including other guidance elements.

18.3.13 Template

A Template is a specific type of guidance that provides for a work product a predefined table of contents, sections, packages, and/or headings, a standardized format, as well as descriptions how the sections and packages are supposed to be used and completed. Templates can be provided for documents and also for conceptual models or physical data stores.

18.3.14 Term Definition

Term Definitions define concepts and are used to build up the Glossary. They are not directly related to Content Elements, but their relationship is derived when the Term is used in the Content Elements description text.

18.3.15 Tool Mentor

A Tool Mentor is a specific type of guidance that shows how to use a specific tool to accomplish some piece of work either in the context of or independent from a Task or Activity.

18.3.16 Whitepaper

Whitepapers are a special Concept guidance that have been externally reviewed or published and can be read and understood in isolation from other content elements and guidance.

18.4 Work Product Kinds

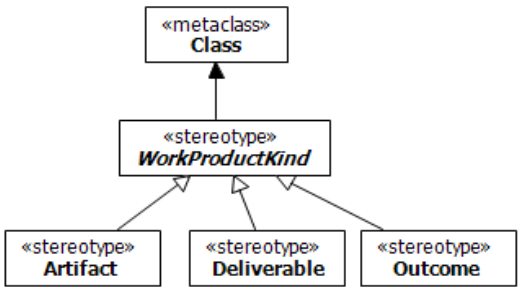


Figure 18.6 - The Work Product Kinds Stereotypes

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
WorkProductKind	Class	n/a	n/a	yes	n/a

18.4.1 Outcome

Outcome Definition is a Work Product Definition that provides a description and definition for non-tangible work products. An outcome describes intangible work products that are a result or state. Outcomes may also be used to describe work products that are not formally defined. A key differentiator of outcomes versus artifacts is that outcomes are not candidates for harvesting as reusable assets.

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
Outcome	WorkProductKind	outcome	n/a	no	

18.4.2 Deliverable

A Deliverable Definition is a Work Product Definition that provides a description and definition for packaging other Work Products, and may be delivered to an internal or external party. Therefore, a Deliverable aggregates other Work Products.

A Deliverable is used to pre-define typical or recommended content in the form of work products that would be packaged for delivery. The packaging of the Deliverable in a process or project could be a modification of this recommendation. Deliverables are used to represent an output from a process that has value, material or otherwise, to a client, customer, or other stakeholder. A Deliverable is a work product that aggregates other work products. Method content maintains preconfigured potential deliverables.

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
Deliverable	WorkProductKind	deliverable	n/a	no	

18.4.3 Artifact

Artifact Definition is a Work Product Definition that provides a description and definition for tangible work product types. Artifacts may be composed of other artifacts. For example, a model artifact can be composed of model elements, which are also artifacts.

Artifacts are tangible work products consumed, produced, or modified by Tasks. They may serve as a basis for defining reusable assets. Roles use Artifacts to perform Tasks and produce Artifacts in the course of performing Tasks. Artifacts are the responsibility of a single Role, making responsibility easy to identify and understand, and promoting the idea that every piece of information produced in the method requires the appropriate set of skills. Even though one role might "own" a specific type of Artifact, other roles can still use the Artifacts, and perhaps even update them if the Role has been given permission to do so.

SPEM 2.0 Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
Artifcat	WorkProductKind	artifact	n/a	no	

18.5 Work Product Relationship Kinds

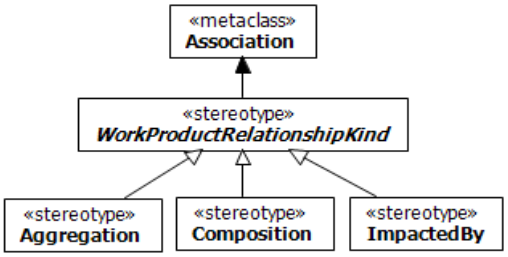


Figure 18.7 - The Work Product Relationship Kinds

Work Product Relationship Kinds define relationships among work products. Typical Kinds are ‘composition’ expressing that a work product use instance of an instance is part of another work product instance of an instance. For example, an instance of Actor is part of an instance of Use Case Model. In contrast to composition, another Kind could express ‘aggregation’ indicating that a Work Product Use is used with another Work Product Use. For example, a customer design deliverable could be defined as a compilation of different other work product uses that are assemble as a report that is delivered to the customer for review. A third key Kind is ‘impacted by’ indicating that a work product use impacts another work product use. For example, if a use case model work product changes, the use case realization work product needs to be updated with these changes.

Profile Notation

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
WorkProductRelationshipKind	association	n/a	n/a	yes	n/a
Composition	WorkProductRelationshipKind	composition	n/a	no	n/a
Aggregation	WorkProductRelationshipKind	aggregation	n/a	no	n/a
ImpactedBy	WorkProductRelationshipKind	impacted by	n/a	no	n/a

Annex A: SPEM 2.0 UML 2 Profile Summary

(normative)

The following table provides a summary of all UML 2 stereotypes defined in the main part of this specification.

Stereotype	Meta-/Superclass	Keyword	Properties	Abstract	Icon
Activity	WorkDefinition, Planned Element / Action	activity	n/a	no	SPEM 1.1 backwards compatibility icon:
AggregatedRole	Association	aggregated role	n/a	no	n/a
BaseConfiguration	/ Dependency	base configuration	n/a	no	n/a
BasePlugin	/ Dependency	base plugin	n/a	no	n/a
BreakdownElement	ProcessElement	n/a	hasMultiple Occurences, isOptional, isPlanned	yes	n/a
Category	DescribableElement / Class	category	n/a	no	
CompositeRole	RoleUse	composite role	n/a	no	
DescribableElement	ExtensibleElement	n/a	presentation Name, brief Description, purpose, main Description	yes	n/a
ExtensibleElement	Classifier	extensible element	kind	yes	

Guidance	DescribableElement / Class	guidance	n/a	no	SPEM 1.1 backwards compatibility icon:
MethodConfiguration	/ Class	configuration	n/a	no	n/a
MethodContentElement	DescribabaleElement	n/a	n/a	yes	
MethodContentPackage	Package	method content package	n/a	no	
MethodContentTrace	/ Dependency	content trace	n/a	no	
MethodLibrary	/ Package	library	n/a	no	
MethodPlugin	/ Package	method plugin	n/a	no	
Metric	Guidance	metric	constraint	no	
Milestone	WorkBreakdownElement / Classifier	milestone	n/a	no	
NestedBreakdownElement	Association	nesting	n/a	no	n/a
OptionalityMandatory	Association	mandatory	n/a	no	n/a
OptionalityOptional	Association	optional	n/a	no	n/a
PackageSelection	/ Dependency	package selection	n/a	no	n/a
ParameterIn	Association	input	n/a	no	n/a
ParameterInOut	Association	inoutput	n/a	no	n/a
ParameterOut	Association	output	n/a	no	n/a
Performer	ExtensibleElement / Association	performs	n/a	no	n/a
PlannedElement	n/a	n/a	startDate, finishDate, rank, duration	yes	n/a
Process	Activity	process	n/a	no	

ProcessComponent	/ Package, Class	process component	n/a	no	SPEM 1.1 backwards compatibility icon:
ProcessComponentUse	BreakdownElement / Classifier	process component use	n/a	no	n/a
ProcessElement	BreakdownElement	n/a	n/a	yes	
ProcessPackage	Package	process package	n/a	no	
Qualification	MethodContent Element / Class	qualification	n/a	no	n/a
RequiredResults	Association	required results	n/a	no	n/a
ResponsibilityAssignment	ExtensibleElement / Association	responsible	n/a	no	n/a
RoleDefinition	MethodContent Element / Class	role definition	n/a	no	SPEM 1.1 backwards compatibility icon:
RoleUse	BreakdownElement / Classifier	role use	n/a	no	SPEM 1.1 backwards compatibility icon:
Step	MethodContentElement, WorkDefinition	step	n/a	no	
Suppressed	Dependency	suppressed	n/a	no	n/a

TaskDefinition	MethodContentElement, WorkDefinition	task definition	n/a	no	SPEM 1.1 backwards compatibility icon:
TaskUse	WorkBreakdownElement, PlannedElement / Classifier, Action	task use	n/a	no	
TeamProfile	BreakdownElement / Classifier	team profile	n/a	no	
ToolDefinition	MethodContent Element / Class	tool definition	n/a	no	
UsedActivity (extension)	Dependency	activity extension	n/a	no	n/a
UsedActivity (localContribution)	Dependency	local contribution	n/a	no	n/a
UsedActivity (localReplace)	Dependency	local replacement	n/a	no	n/a
VariabilityContributes	Variability Specialization	contributes	n/a	no	n/a
VariabilityExendsReplaces	VariabilitySpecialization	extends-replaces	n/a	no	n/a
VariabilityExtends	VariabilitySpecialization	extends	n/a	no	n/a
VariabilityReplaces	VariabilitySpecialization	replaces	n/a	no	n/a
VariabilitySpecialization	Generalization	n/a	n/a	yes	n/a
WorkBreakdownElement	BreakdownElement	n/a	isReatable, isOngoing, isEventDriven	yes	n/a
WorkDefiniton	Classifier, Action	n/a	precondition, postcondition	yes	n/a
WorkProductDefinition	MethodContent Element / Class	work product definition	n/a	no	
WorkProductPort	ProcessElement / Port	work product port	portKind, isOptional	no	n/a

WorkProductRelationship	ExtensibleElement / Association	related work product	n/a	no	n/a
WorkProductUse	BreakdownElement / Classifier	work product use	n/a	no	
WorkSequence	BreakdownElement / Association	predecessor	linkKind	no	n/a

Annex B: Additional Variability Examples

(normative)

This annex provides additional background and examples for the Variability Types defined in Section 14.11.

B.1 General Principles

The rules defined in the following sections are all derived from these three simple principles:

- Contribution: Only adds attributes and associations to the base. It never overrides or replaces any information from the base, i.e., if the base is allowed an association to exactly one other element and has this one already defined, it is not replaced by a contributors association.
- Replace: Always replaces attributes and associations of the base element with the replacing elements attributes and associations, except for incoming associations, which are only added to the base but do not replace the base's incoming associations.
- Extends: Only defines inheritance for the extending element. The base remains untouched. The extending element can override inherited attribute values and association instances by defining their own. If no new values are defined for the extending element, it will inherit these values from the base element.

B.2 Contributes

Background

Contributes provides a way for instances of Variability Elements to contribute their properties into their base Variability Element without directly altering any of its existing properties, i.e., in an additive fashion. Properties contributed are: attribute values and association instances. The effect after interpretation of contribution is that the base Variability Element is logically replaced with an augmented variant of the element that combines attribute values and association instances. The way this combination is realized depends on the type of the attribute or association. For example, String attributes are concatenated resolving embedded commands for dependent text or merging text fragments (e.g., descriptions for content elements). Additional elements in to-many associations are added (e.g., additional Guidance elements or Task Uses of an Activity). Different elements in to-one associations are ignored (e.g., the one Primary Performer of a Task). Multiple Variability Elements can contribute to the same base element and all of these contributions properties are added to the base in the same fashion.

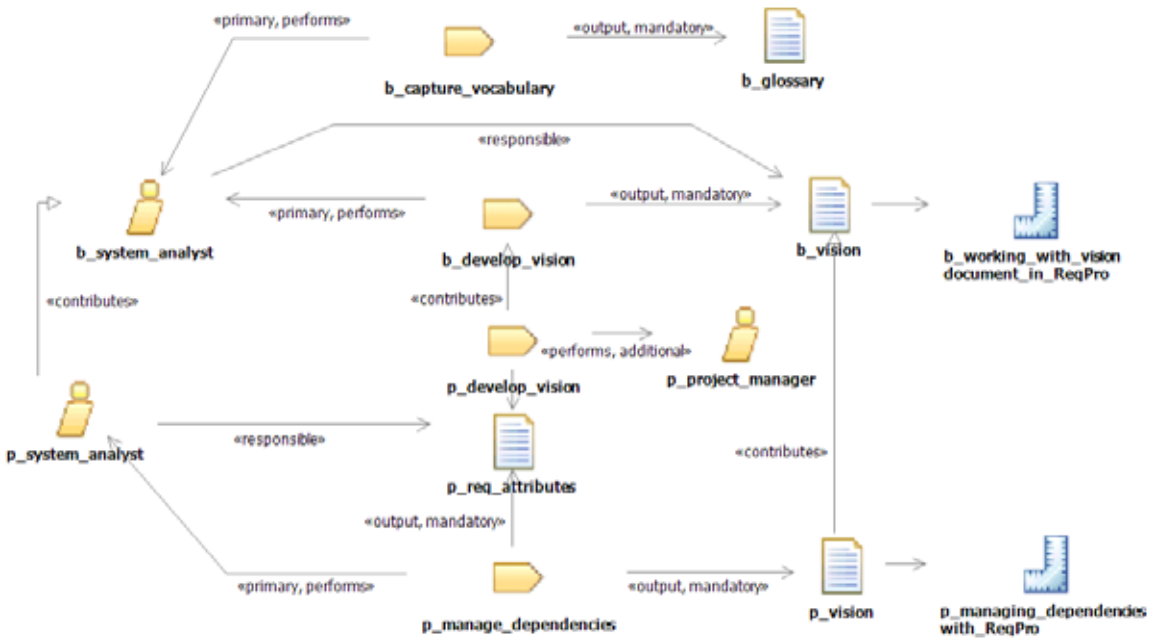
Rules

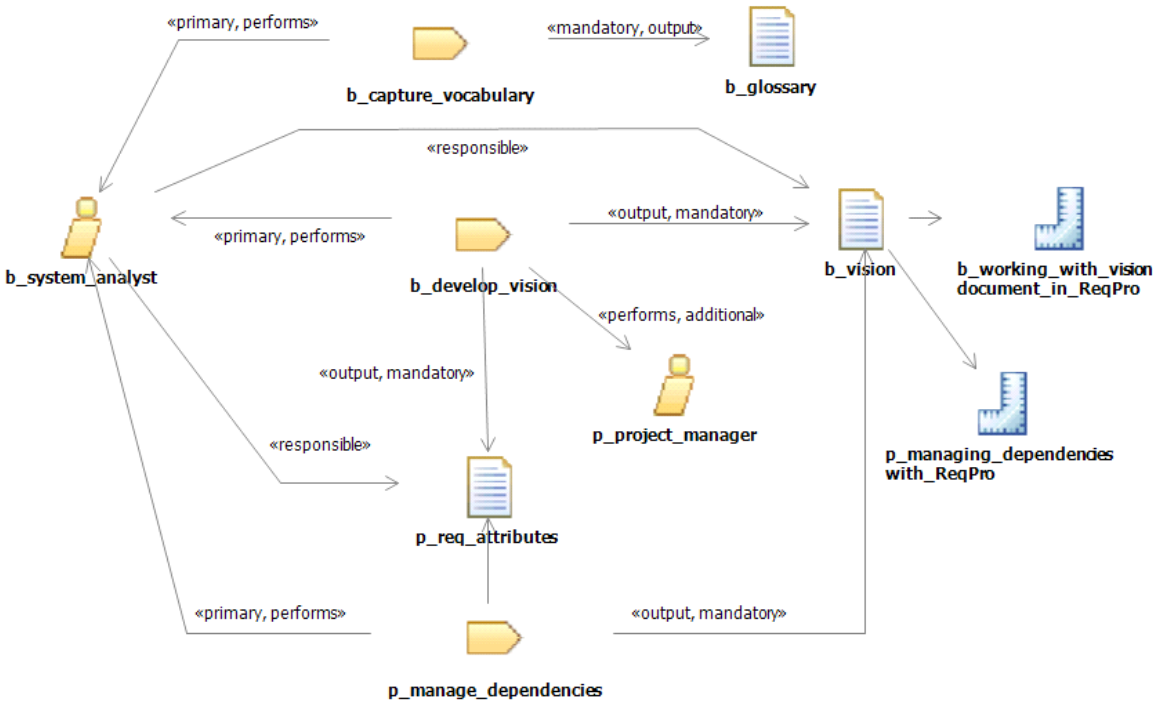
- Attributes are appended except for identifying/naming attributes and non-string attributes such as boolean, date, etc.
- Outgoing to-one associations of the contributor will be ignored if the base already has such an association or if the base element has more than one contributing element (i.e., it cannot be decided which contribution should be chosen, then non will be chosen).
- Outgoing to-many associations are added to the base elements of a contribution.

- Incoming associations to the contributing element with a to-one constraint on the other association end will be ignored if the base already has such an incoming association or if there is more than one contribution to consider for the base element.
- Other incoming associations to the contributing element are added to the base elements of a contribution.
- A base element of a contribution can have more than one contributor.
- Contribution works transitively, i.e., a contribution element contributes its own contributors.

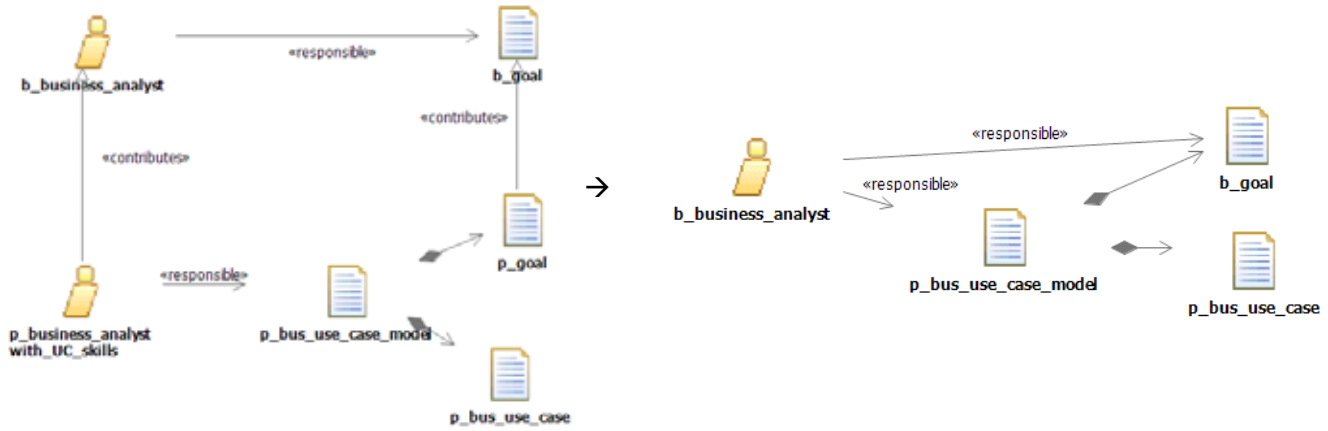
Examples

- This example shows three contributions and the overall result when the relationships are interpreted and published. Note that the primary performer association between p_develop_vision and p_system_analyst is not needed, but if it is modeled, then the contribution interpretation will ignore the second association:

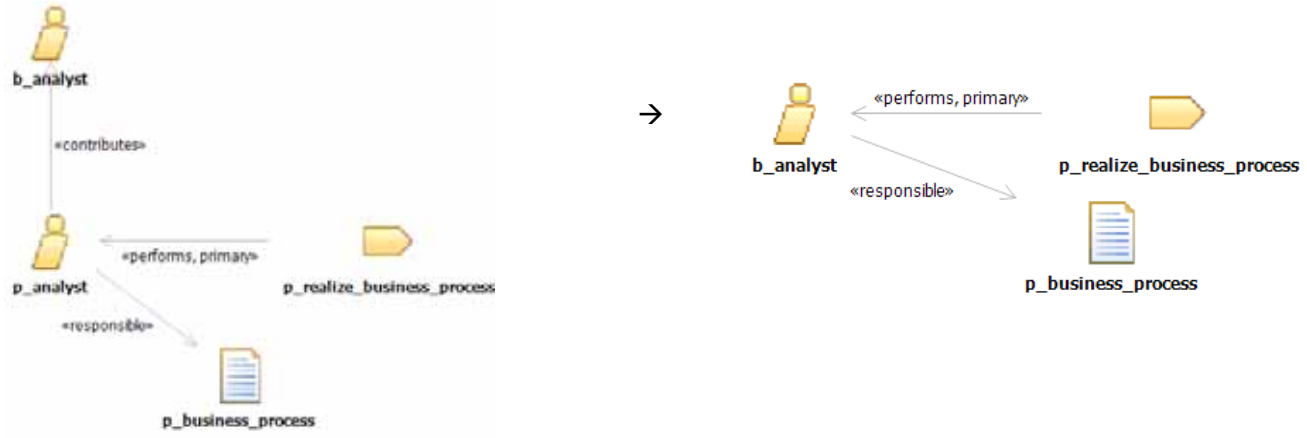




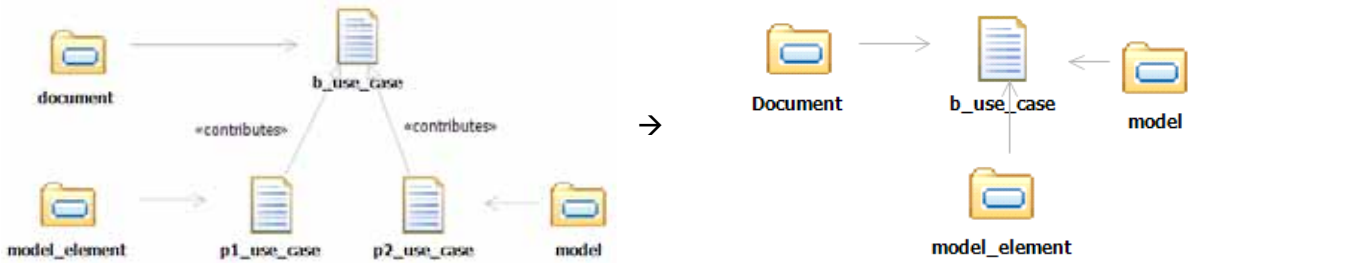
Example for incoming association: if contributor is part of a composition, then the base element also will be part of this composition:



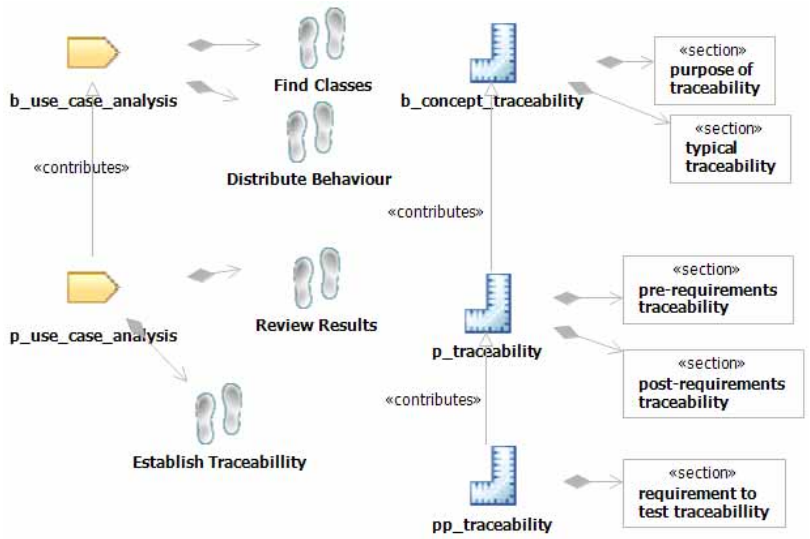
Second example for incoming associations (as well as outgoing association):



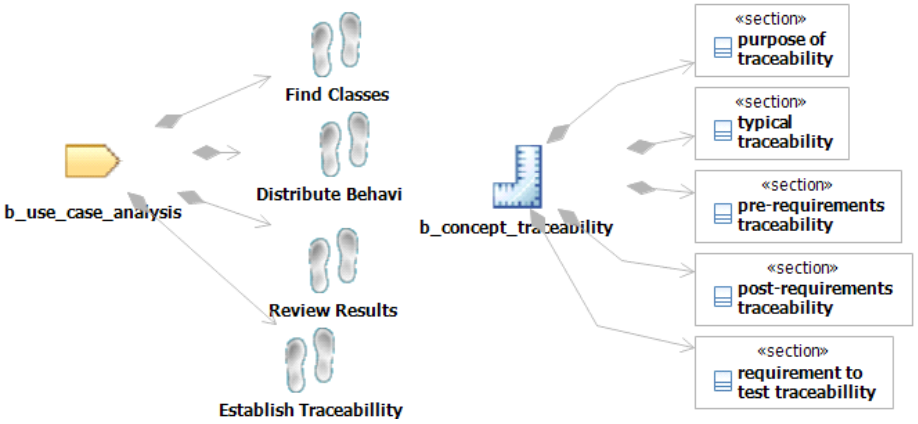
This example shows an incoming association that has a to-many constraint on the other association end (i.e., a work product can have many content categories such as work product kinds). As a result, contributions are added to the base:



This example shows that Sections of Describable Elements can also be contributed to base elements:


b_use_case_analysis
Find Classes
Distribute Behaviour
«contributes»
p_use_case_analysis
Review Results
Establish Traceabillity
b_concept_traceability
«section»
purpose of traceability
«section»
typical traceability
«contributes»
p_traceability
«section»
pre-requirements traceability
«section»
post-requirements traceability
«contributes»
pp_traceability
«section»
requirement to test traceabillity

→


Find Classes
b_use_case_analysis
Distribute Behavi
Review Results
Establish Traceabillity
b_concept_traceability
«section»
purpose of traceability
«section»
typical traceability
«section»
pre-requirements traceability
«section»
post-requirements traceability
«section»
requirement to test traceabillity

B.3 Replace

Background

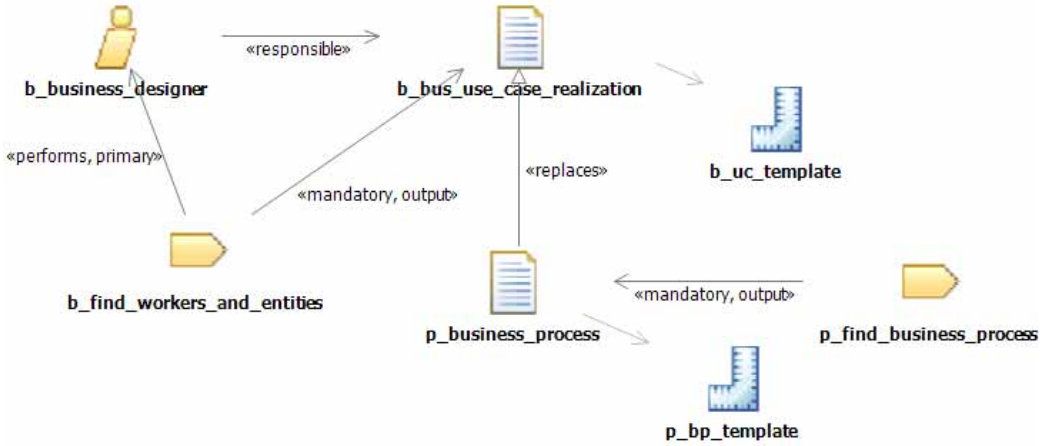
Replaces provides a way for Variability Elements to replace a base Variability Element without directly changing any of its existing properties. This is in most cases used for Method Plugins that aim to replace specific Content Elements such as Roles, Task, or Activities with either a complete new variant or to change fundamental relationships of these elements (e.g., Role-Artifact responsibility). Properties replaced are attribute values and association instances. The effect of this is that the base Content Element is logically replaced with this new variant of the element to which all incoming associations still point as before, but which has potentially new attribute values and outgoing association properties. This provides a very powerful mechanism to replace, for example, whole Activities in a Process with complete new realizations of the Activity. For instance, replacing an Activity doing use case-based design with an activity doing agile code-centric development doing the same work using a different development technique utilizing different Roles, Tasks, etc. Another example would be to replace an Activity that describes database design for an RDBMS with an Activity that describes database design for an OODBMS. A Variability Element can only be replaced by one other element at a time. For example, if two Method Plugins replace the same element, only one Method Plugin can be used for interpretation.

Rules

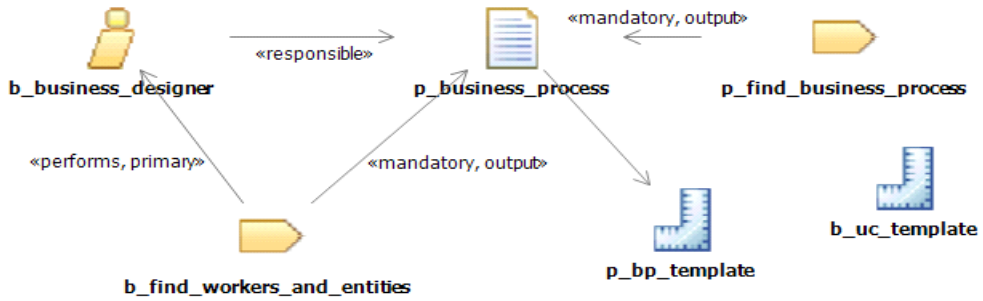
- Attributes of the base element are replaced with attributes of the replacing element, including identifiers of the base element.
- Outgoing to-one and to-many associations of the replacing element replace any outgoing associations of the base element. If the contributor does not have any outgoing associations, then the resulting element will also not have any.
- Incoming association with a to-many constraint on the other association end to the replaced element will be preserved and augmented with incoming associations of the replacing element.
- Incoming associations with a to-one constraint on the other association end to the replaced element will be replaced with the respective incoming association of the replacing element. If the replacing element does not have such an association, then the resulting element will also keep the incoming association of the base element.
- A base element of a replacement can have only one replacing element per configuration. If more than one replacing element is present, no replacement at all will take place.
- Replacement works transitively, i.e., if a replacing element is replaced itself, the final replacer will prevail.
- Contribution precedes Replacement, i.e., contribution is evaluated first and then replacement performed afterwards. The evaluation of contribution and replacement is performed top-down in the specialization hierarchy.

Examples

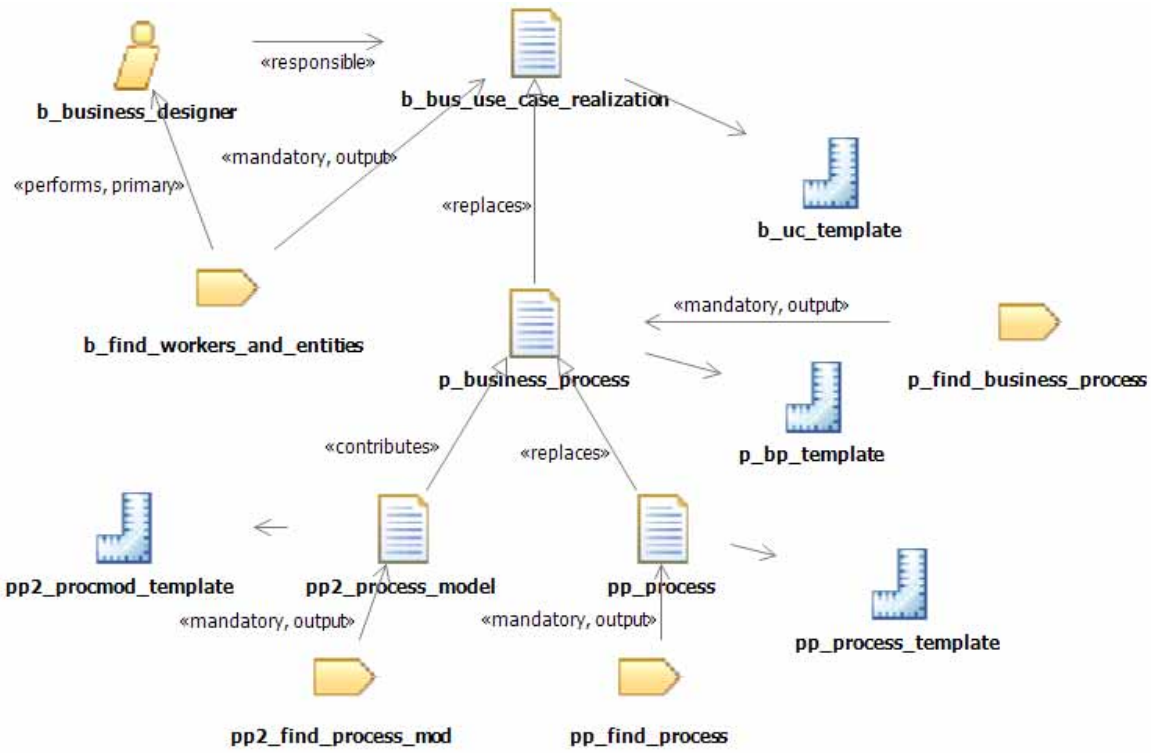
This example shows how a work product is replaced and its associations reinterpreted: Incoming to-many associations to both the base and replacing element are preserved. Outgoing associations are replaced.



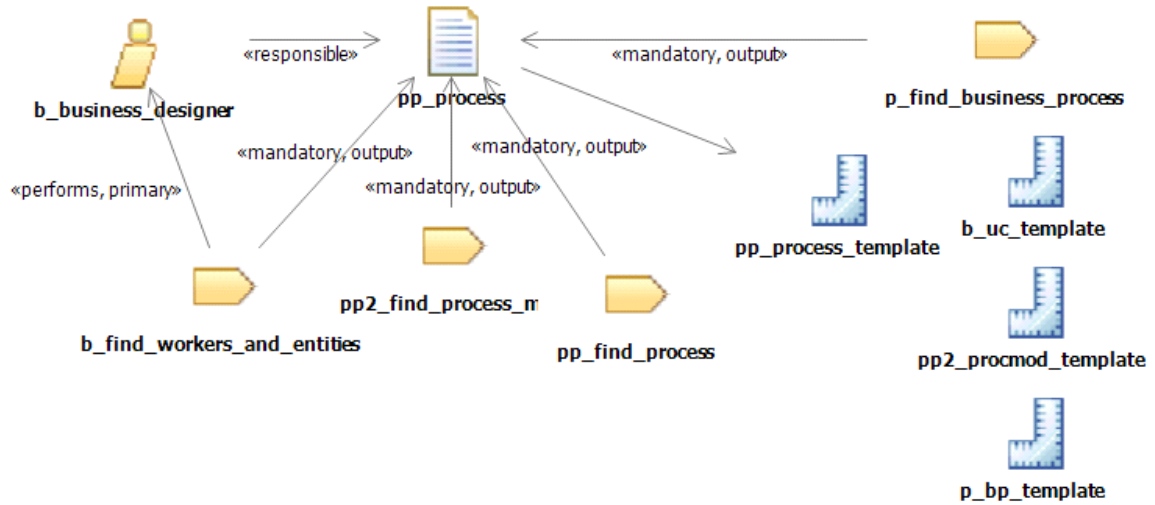
→



This example shows transitive definition of replacement plus the combination of contribution with replace. The work product hierarchy relationships are evaluated top-down and then contribution first replacement second. Hence, p_business_process replaces b_bus_use_case_realization first, then the contribution of pp2_process_model are added and then everything is replaced with pp_process. The result shows that incoming associations from Tasks are preserved at that only the outgoing association to the pp_process_template guideline is preserved, because of the final replace of pp_process.


«responsible»
b_business_designer
b_bus_use_case_realization
«mandatory, output»
«performs, primary»
«replaces»
b_uc_template
b_find_workers_and_entities
«mandatory, output»
p_business_process
p_find_business_process
p_bp_template
«contributes»
«replaces»
pp2_procmod_template
pp2_process_model
pp_process
«mandatory, output»
«mandatory, output»
pp_process_template
pp2_find_process_mod
pp_find_process

→


«responsible»
«mandatory, output»
b_business_designer
pp_process
p_find_business_process
«mandatory, output»
«mandatory, output»
«performs, primary»
«mandatory, output»
pp_process_template
b_uc_template
pp2_find_process_m
b_find_workers_and_entities
pp_find_process
pp2_procmod_template
p_bp_template

B.4 Extends

Background

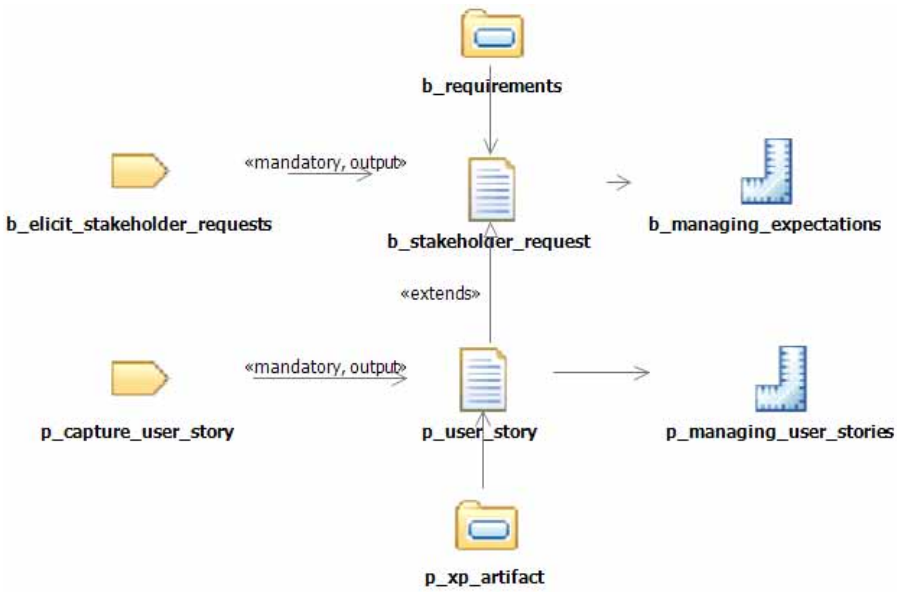
Extension allows Method Plugins to easily reuse elements from a Base Plugin by providing a kind of inheritance for the special Variability Element. Attribute values and Association instances are inherited from the based-on Variability Element to the special Variability Element. The result of interpretation is that the special element has the same properties that the based-on has, but might define its own overrides. Hence, extends is not used to modify content of the base Plugin, but to provide the ability for the extending Plugin to define its own content which is a variant of content already defined (e.g., a special version of a generic Review Record for a specific type of review). The effect of this is that the base Variability Element and any number of extending Variability Elements can be used side by side, but refer to each other via the extends relationship. Extends also provides the key mechanism for binding Process Patterns to Processes: A pattern is applied by defining an extends relationships from an Activity of the applying Processes to the Pattern. The Activity inherits association instances from the pattern and the pattern appears to be part of the resulting Process after Interpretation.

Rules

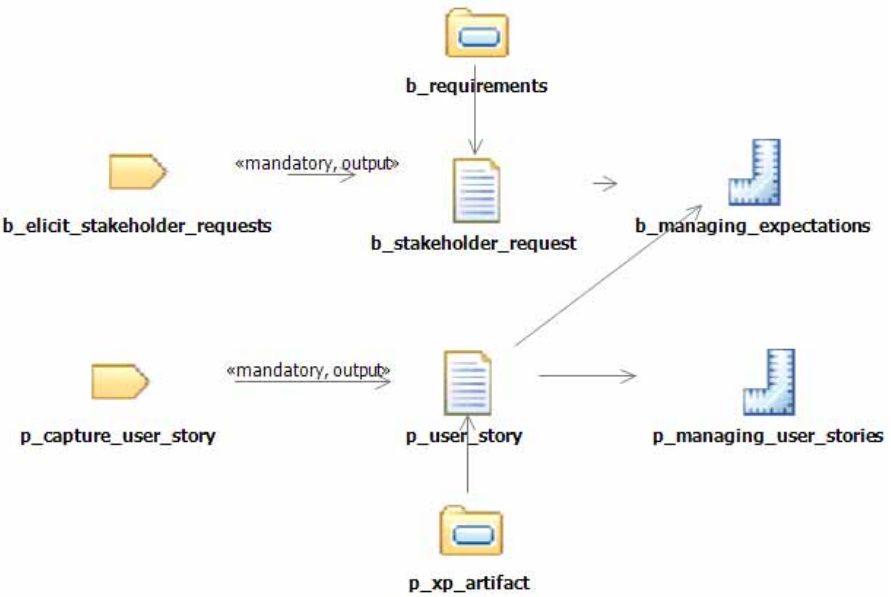
- Attribute values of the base element are inherited to the extending element if the extending element has not defined its own values.
- Outgoing to-many associations of the base element are inherited to the extending element if the extending element does not define its own association instances.
- Outgoing to-one association are only inherited if the extending element does not define its own association instance.
- Extends works transitively, i.e., if an extending element is extended itself the second extension inherits from its direct and indirect base elements.
- Contribution precedes Extends, i.e., contribution is evaluated first and then extending elements inherit afterwards from the base element (including all of its contribution).
- Replace precedes Extends, i.e., if a base element has a replace and extends relationship, the extending element inherits from the replacing element.

Example

The example shows how different associations will be interpreted when an extends relationship is published. p_user_story inherits the relationship to guidance (b_managing_expectations) related from the base element.


b_requirements
«mandatory, output»
b_elicit_stakeholder_requests
b_stakeholder_request
b_managing_expectations
«extends»
«mandatory, output»
p_capture_user_story
p_user_story
p_managing_user_stories
p_xp_artifact

→


b_requirements
«mandatory, output»
b_elicit_stakeholder_requests
b_stakeholder_request
b_managing_expectations
«mandatory, output»
p_capture_user_story
p_user_story
p_managing_user_stories
p_xp_artifact

B.5 Extends-Replaces

Background

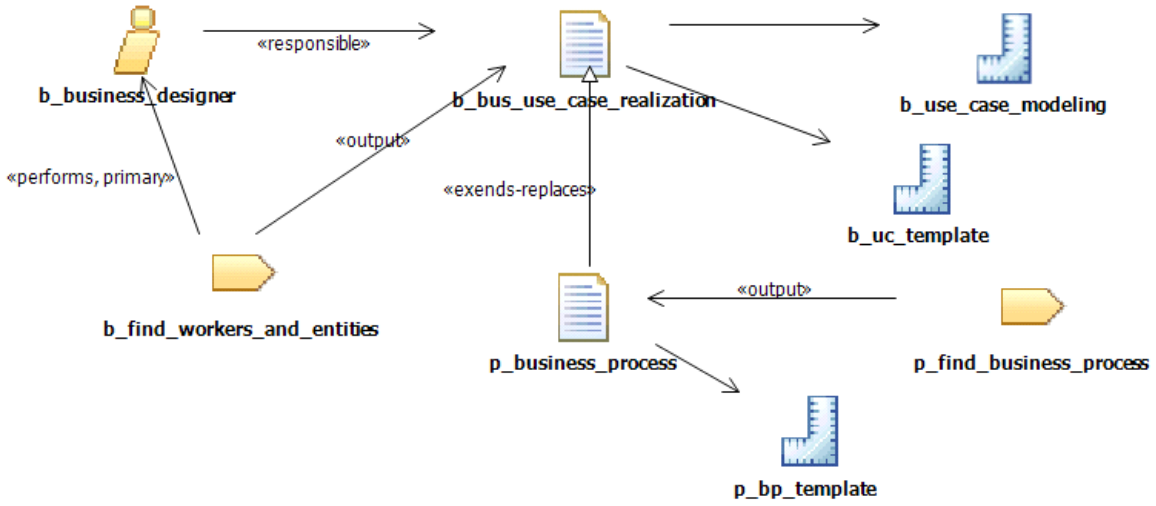
The extends and replaces variability relationship combines the effects of extends and replaces variability into one variability type. Whereas replaces variability completely replaces all attributes and outgoing associations of the base element with new values and instances, or removes all values or associations if the replacing element does not define any, extends and replaces variability only replaces values that have been redefined. All other values of the base element are unaffected. In other words, extends and replaces allows users to selectively replace specific attributes and associations of the base elements. This type of variability can be used to generate method plug-ins that rename elements, or replace some descriptions of method elements with new ones, without completely remodeling all other relationships and attributes needed by the base plug-in.

Rules

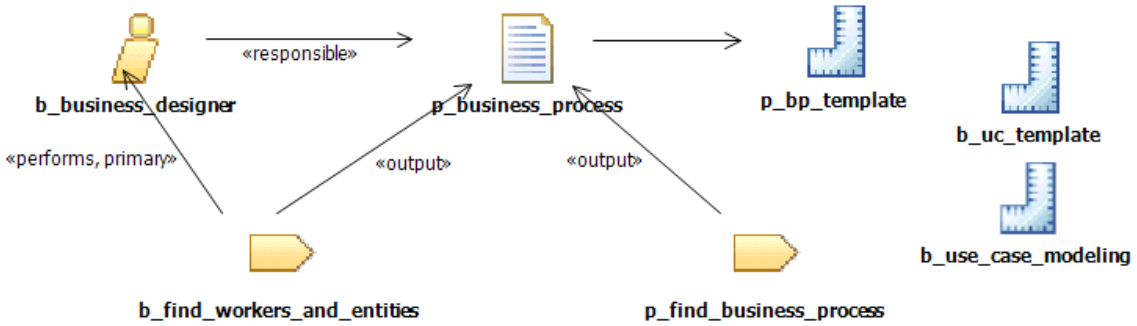
- Extends and replace variability combines the effects of the extends and the replaces variability. The evaluation will first perform the effects of the extends and then the effects of the replaces variability. This implies the following:
  - First, the new element will inherit all attributes and associations from the base element.
  - Second, the new elements might override inherited attributes or associations.
  - Third, the base element will be replaced with new element using the overridden values and if no override was specified keeping the inherited values.
- If the extends and replaces element defines outgoing associations, they will replace all outgoing associations of the base elements. If the extends and replaces element does not define any new associations, the resulting element will retain the associations of the base element.
- Incoming associations from the base element are added to the replacing element.
- If the extends and replaces element defines attributes, these attributes are replaced in the resulting element including the base element's identifier. Undefined attributes retain values used in the base element.
- The base element of a replaces relationship or an extends and replaces relationship can have only one replaces or extends and replaces element per configuration. If more than one element is present, no replacement takes place.
- The extends and replaces relationship is transitive and evaluated top-down relative to the direction of the replacement. If a replacing element is also replaced, the final replacing element prevails.
- Contributes variability relationships are resolved before replaces and extends and replaces relationships. Extends relationships are resolved last. Variability is always resolved top-down from the base to the variability elements. Within the same level, contributes relationships are resolved first. Replaces or extends and replaces are resolved afterwards.

Example

The example below shows how extends-replace works by having the extends-replacing element p_business_process inherit is base's relationships, but overriding the relationships to guidance elements as part of the replacement.


«responsible»
b_business_designer
b_bus_use_case_realization
b_use_case_modeling
«output»
«performs, primary»
«exends-replaces»
b_uc_template
b_find_workers_and_entities
«output»
p_business_process
p_find_business_process
p_bp_template

→


«responsible»
b_business_designer
p_business_process
p_bp_template
b_uc_template
«performs, primary»
«output»
«output»
b_use_case_modeling
b_find_workers_and_entities
p_find_business_process

Annex C: Case Studies and Examples

(normative)

This annex provides a set of case studies and examples of processes that have been already successfully modeled or re-modeled with SPEM 2.0 concepts. These case studies have been used to validate and verify the SPEM 2.0 concepts defined in this specification.

C.1 Fujitsu DMR Macroscope

This first example has been directly ported from the SPEM 1.1 specification (formal/05-01-06, Appendix C) as a case study how SPEM 1.1 processes can be easily migrated into the SPEM 2.0 format. This example, taken from the Fujitsu DMR Macroscope method, only represents a portion of a typical information system delivery process. Process meta-model (M2) classes, associations and attributes are represented in `courier` while the corresponding M1 instances appear in **bold times** font.

`Activity {kind: Phase}`: **Preliminary Analysis**
`Process`: **Information System Delivery Process**

`Activity {kind: Iteration}`: **First Joint Requirements Planning (JRP) Workshop**
`TaskUse`: **Define Owner Requirements**
`ProcessPerformer {kind: primary}`
`RoleUse`: **System Architect** {`kind` : in}
`WorkDefinitionParameter` {`kind` : in}
`WorkProductUse`: **EnterpriseArchitecture**
`WorkDefinitionParameter` {`kind` : out}
`WorkProductUse`: **Assessment of Current System {**`state`**: initial draft}**
`WorkProductUse`: **Owner Requirements {**`state`**: initial draft }**
`Steps`
`Step` : **Define objectives based on stated needs**
`Step` : **Define the key issues**
`Step` : **Determine the relevant enterprise principles**

`TaskUse`: **Draft Owner Models**
`ProcessPerformer {kind: primary}`
`RoleUse`: **System Architect**
`WorkDefinitionParameter` {`kind` : in}
`WorkProductUse`: **Assessment of Current System {**`state`**: initial draft }**
`WorkProductUse`: **Owner Requirements {**`state`**: initial draft }**
`WorkDefinitionParameter` {`kind` : out}
`WorkProductUse`: **Business Structure {**`state`**: initial draft }**
`WorkProductUse`: **Business Dynamics {**`state`**: initial draft }**
`Steps`

Step: **Determine System context**
Step: **Model structural and dynamic aspects of the enterprise**
Step: **Define work resources**
Step: **Explore with prototypes**

TaskUse: **Define User Requirements**
ProcessPerformer {kind: primary}
RoleUse: **System Architect**
WorkDefinitionParameter {kind : in}
WorkProductUse: **Assessment of Current System {**state**: initial draft }**
WorkProductUse: **Owner Requirements {**state**: initial draft }**
WorkDefinitionParameter {kind : out}
WorkProduct : **User Alternatives {**state**: initial draft }**
WorkProduct : **User Principles {**state**: initial draft }**
Steps
Step: **Consider user interface aspects**
Step: **Consider distribution aspects**
Step: **Explore with prototypes**

TaskUse: **Draft User Models**
ProcessPerformer {kind: primary}
RoleUse: **System Architect**
WorkDefinitionParameter {kind: in}
WorkProductUse: **User Alternatives {**state**: initial draft }**
WorkProductUse: **User Principles {**state**: initial draft }**
WorkProductUse: **Business Structure {**state**: initial draft }**
WorkProductUse: **Business Dynamics {**state**: initial draft }**
WorkDefinitionParameter {kind: out}
WorkProductUse: **System Structure {**state**: initial draft }**
WorkProductUse: **System Dynamics {**state**: initial draft }**
Steps
Step : **Determine System context**
Step : **Model structural and dynamic aspects of the system**
Step : **Define work resources**
Step : **Explore with prototypes**

TaskUse: **Define Developer Requirements**
ProcessPerformer {kind: primary}
RoleUse: **Technical Architect**
WorkDefinitionParameter {kind: in}
WorkProductUse: **User Alternatives {**state**: initial draft }**
WorkProductUse: **User Principles {**state**: initial draft }**
WorkDefinitionParameter {kind : out}
WorkProductUse: **Developer Alternatives {**state**: initial draft }**
WorkProductUse: **Developer Principles {**state**: initial draft }**
WorkProductUse: **Technology Infrastructure {**state**: initial draft }**

Steps
Step: **Revise work process and class definitions**
Step: **Revise user interface models**

TaskUse: **Draft Developer Models**
ProcessPerformer {kind: primary}
RoleUse: **Technical Architect**
WorkDefinitionParameter {kind : in}
WorkProductUse: **Developer Alternatives {**state**: initial draft }**
WorkProductUse: **Developer Principles {**state**: initial draft }**
WorkProductUse: **Technology Infrastructure {**state**: initial draft }**
WorkProductUse: **System Structure {**state**: initial draft }**
WorkProductUse: **System Dynamics {**state**: initial draft }**
WorkDefinitionParameter {kind : out}
WorkProductUse: **Software Architecture {**state**: initial draft }**
WorkProductUse: **Persistent Information {**state**: initial draft }**
Steps
Step : **Define process and data aspects of the system**
Step : **Consider user interface aspects**
Step : **Consider distribution aspects**
Step : **Explore with prototypes**

Activity (kind: Iteration): **Second Joint Requirements Planning (JRP) Workshop**
TaskUse**…**
Similar to **First Joint Requirements Planning (JRP) Workshop** iteration:

- reuse and cumulate existing WorkProductUse assets as input to activities
- change «**initial draft** » output WorkProductUse states with «**revised draft** »

Activity (kind: Phase): **System Architecture**
Process: **Information System Delivery Process**

Activity (kind: Iteration): **First Joint Application Design (JAD) Workshop**
TaskUse: **Revise User Models**
ProcessRole : **System Architect**
ActivityParameters {kind : input}
WorkProduct : **System Structure {**state**: revised draft }**
WorkProduct : **System Dynamics {**state**: revised draft }**
ActivityParameters {kind : output}
WorkProduct : **System Structure {**state**: revised }**
WorkProduct : **System Dynamics {**state**: revised }**
Steps
Step : **Revise work process and class definitions**
Step : **Revise user interface models**
Step : **Realize/improve prototype**
etc.

`Activity (kind: Phase)`: **System Architecture**

etc.

C.2 Microsoft Solution Framework Agile Case Study

This case study is a proof of concept showing how Microsoft's publicly available Team System process "Microsoft Solution Framework (MSF) Agile" can be represented using SPEM 2.0 concepts. The aim of the proof of concept was to show that SPEM 2.0's concepts are sufficient to model all of MSF Agile's process concepts and structures; in other words the MSF Agile meta-model can be projected onto the SPEM 2.0 meta-model.

The following table of Figure C.1 shows the conceptual mapping of the MSF Agile concepts to SPEM 2.0 concepts. It further shows the parts of MSF Agile that have been modeled exemplarily with these SPEM 2.0 concepts. The full case study is available on request from Peter Haumer, phaumer@us.ibm.com.

MSF Agile	EPF (SPEM 2.0)	Examples available
Role	Role	Role: Project Manager, Role: Architect
Work Item	Artifact	<all, see Work Items>, Artifact: Bug, Artifact: Scenario
Work Product	Artifact	<all in Program Management and Architecture>, Artifact: Iteration Plan, Artifact: Application Diagram
Activity	Task	<all Program Management and Architecture>, Task: Create Infrastructure Architecture, Task: Monitor Iteration
Sub-Activity	Step	<some in Project Management (e.g. in Process Pattern: Guide Iteration workstream) and all in Discipline: Architecture (Process Pattern: Create Solution Architecture workstream)>
Workstream	Activity/Process Pattern	Process Pattern: Capture Project Vision, Process Pattern: Create Solution Architecture, Process Pattern: Guide Iteration, Process Pattern: Plan an Iteration
Track	Process Pattern (Track BreakdownElementKind)	Process Pattern: Envision, Process Pattern: Plan
Discipline	Process Pattern (Discipline BreakdownElementKind)	Process Pattern: Project Management
Cycle	Process Pattern (Cycle BreakdownElementKind)	Process Pattern: Daily Build
Quality of Service	Process Pattern	Process Pattern: User Experience
Report/Queries	Guidance (Report Guidance Kind)	Report: Bugs by Priority, Report: Velocity
How To	Guidance (Guideline GuidanceKind)	Guideline: How to: Create a Bug List, Guideline: How to: Create an Iteration Plan
Procedure	Guidance (Tool Mentor GuidanceKind)	Tool Mentor: Break down scenarios into tasks in Microsoft Project, Tool Mentor: To Create a Bug List
Principle	Guidance (Practice GuidanceKind)	Practice: Flow of value, Practice: Foster open communications

Figure C.1 - Conceptual mapping of MSF Agile to SPEM 2.0 concepts

The SPEM 2.0 models produced for this case study have been deliberately presented in a similar fashion as the original as html published content to allow a side by side comparison. Figure C.2 shows an example of how Roles are presented. Figure C.2, left shows the original MSF Agile presentation of a Role and the Figure C.2, right shows the equivalent modeled with SPEM 2.0 concepts. One can see that all key relationships of the Role have been represented on the right side as well. For example, a Role is linked to a set of Work Products, which has been modeled by the responsibility as

well as modifies association in SPEM. Although MSF Agile does not specify the nature of the Role-Work Product relationship, we modeled these relationships to show how they can be mapped. Responsibility had been modeled directly by connecting roles and work products listed for a Role.

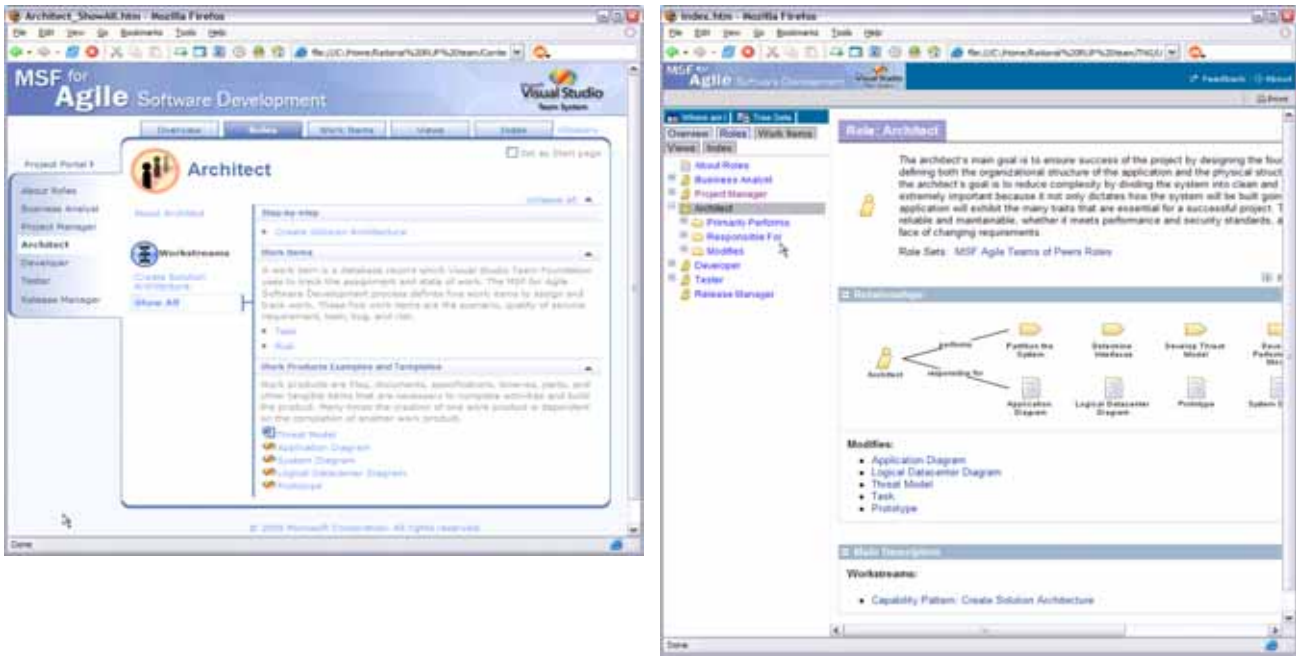


Figure C.2 - Side-by-side comparison of a Role presentation and associated concepts in MF Agile (left) and modeled with SPEM 2.0 concepts (right)

Modifies links are derived links in SPEM 2.0 by evaluating the outputs of Tasks that the Role performs. Although MSF Agile does not model outputs of Activities (SPEM 2.0 Tasks), we interpreted what the output would be by looking at the MSF Agile Activity's Exit Criteria and Activity descriptions. This shows the advantages of SPEM 2.0 which makes this information explicit in the model (in addition to modeling the same structures MSF Agile defines) rather than keeping it informal in the textual descriptions as done by MSF Agile.

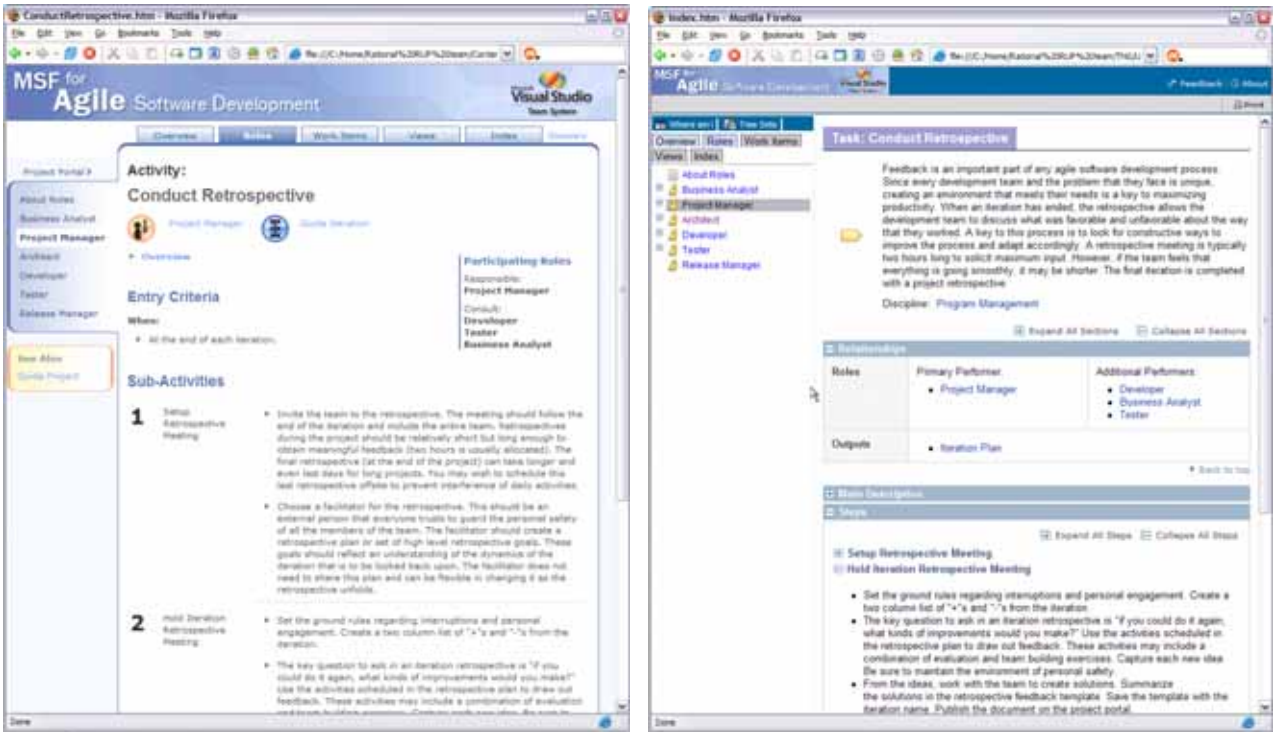


Figure C.3 - Mapping of an MSF Activity to a SPEM 2.0 Task

Thus, Figure C.3 shows an example of representing an MSF Agile Activity as a SPEM 2.0 Task with the additions mentioned above: an explicitly modeled output work product. One can also see in Figure C.3 the one to one mappings of other relationships such as the responsible Role and the consulting Roles of the Activity mapped to primary performer and additional performer in SPEM 2.0. MSF Agile's Sub-Activities have been modeled as Steps in SPEM 2.0.

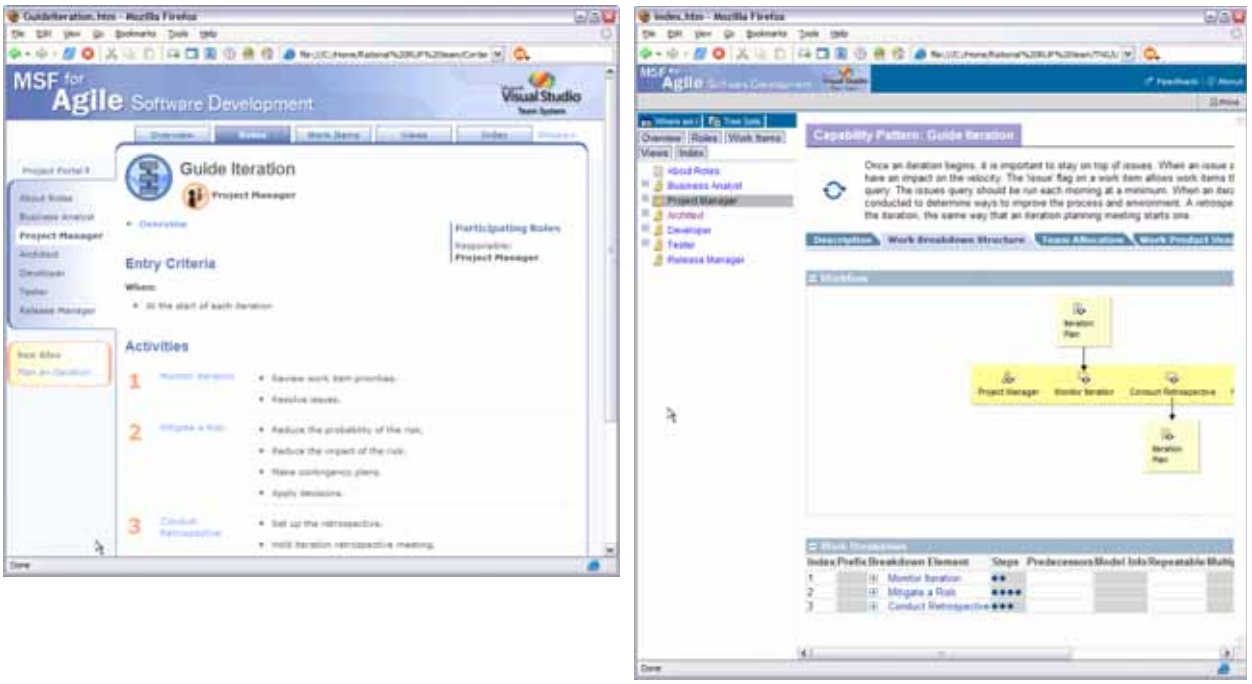


Figure C.4 - Mapping of MSF Agile Workstreams to SPEM 2.0 Process Patterns

We modeled MSF Agile's Workstreams as SPEM 2.0's Process Patterns since both concepts represent the grouping of work into higher level units. As one can see in Figure C.4, both approaches use breakdown structures to represent the concept. In addition to the information modeled in MSF Agile, you see in Figure C.4 a visualization of the input/output relationships of tasks to work products.

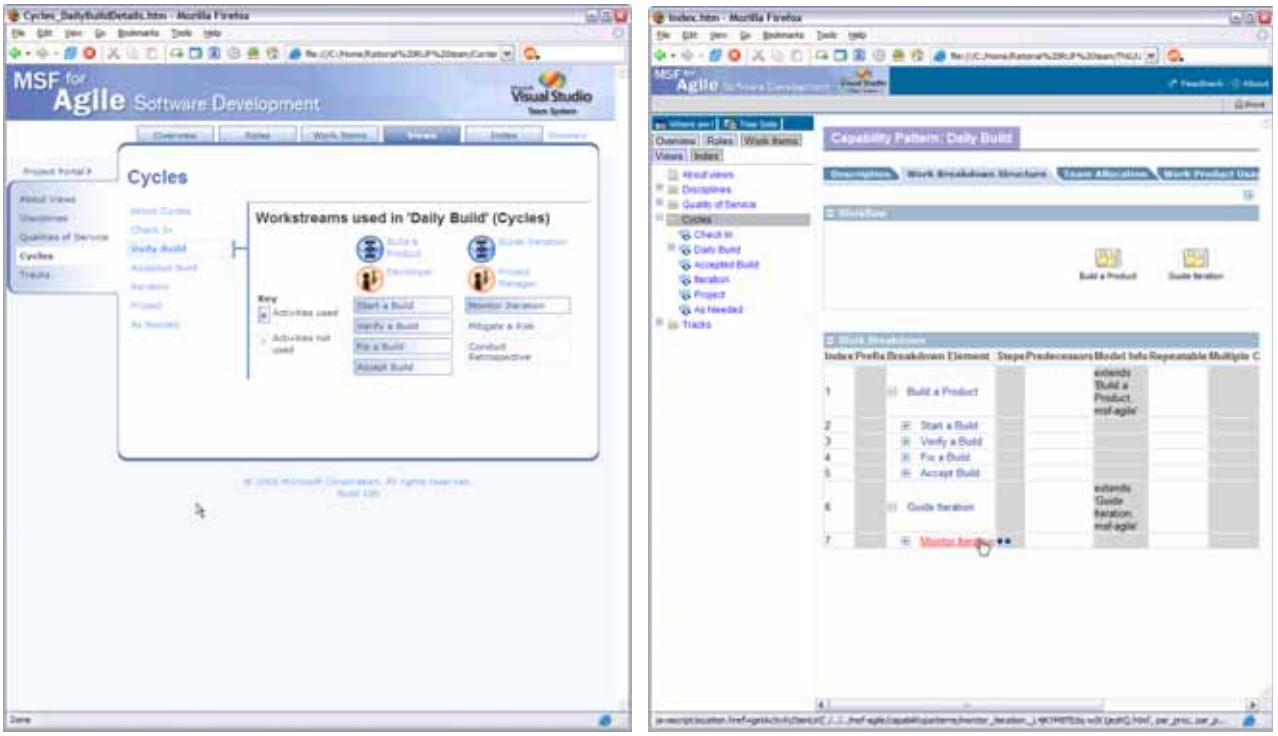


Figure C.5 - Modeling MSF Agile process Views as SPEM 2.0 Process Patterns

Finally, we show in Figure C.5 how MSF Agile's different process views can be represented as SPEM 2.0 Process Patterns as well. Cycle views, such as the "Daily Build" depicted in Figure C.5 is a collection of MSF Agile Workstreams in which specific Activities are not being 'used.' We modeled these as Process Patterns that are composed of other Process Patterns that represent the Workstreams and for which the unused Tasks have been suppressed.

C.3 Eclipse Process Framework OpenUP/Basic

The Eclipse Process Framework (EPF) project (www.eclipse.org/epf) is a new Eclipse Technology open source project that aims to provide an extensible framework and exemplary tools based on SPEM 2.0 concepts for defining and managing software development processes. Within this framework, the project develops extensible process content for a range of software development and management processes supporting iterative, agile, and incremental development, and applicable to a broad set of development platforms and applications. Figure C.6 provides an overview of the process ecosystem EPF aims to provide. It shows how the tool architecture which currently realizes a major subset of the SPEM 2.0 specification accommodates content from various sources and for many different target audiences.

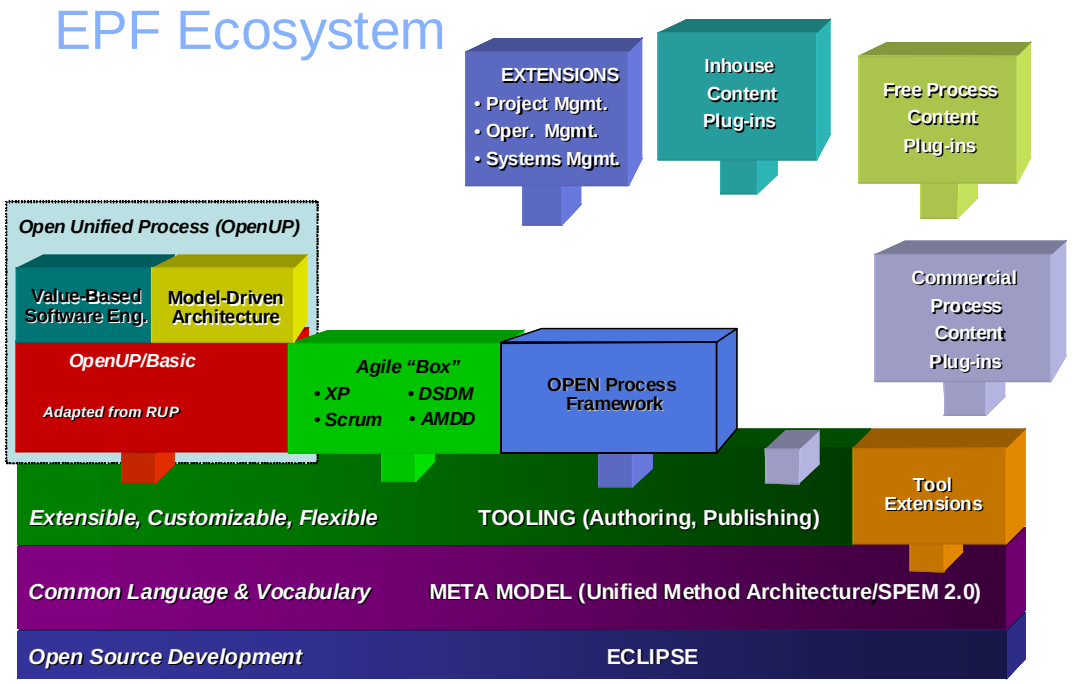


Figure C.6 - Eclipse Process Framework Ecosystem accommodating content from many different sources

A key content package, i.e., Method Plug-in depicted in Figure C.6 is the OpenUP/Basic method framework, which is developed by key committers of the EPF project. OpenUP/Basic is an iterative software development process that is minimal, complete, and extensible.

- The process is minimal in that only fundamental content is included.
- The process is complete in that it can be manifested as an entire process to build a system.
- The process is extensible in that it can be used as a foundation on which process content can be added or tailored as needed.

OpenUP/Basic takes an agile approach to development, valuing team collaboration and benefits to the stakeholders over unproductive deliverables and formality. The process provides this progressive approach to building systems within a proven, structured lifecycle.

OpenUP/Basic is driven by the following core principles:

- Application of an iterative lifecycle that mitigates risk early and often, and shows results early and often
- Focus on the collaboration within a development team including the product stakeholders to maximize results
- Management of requirements in a form that represents stakeholder value and drives the development process
- Cognizance of architecture as a means to increase quality and technical understandability

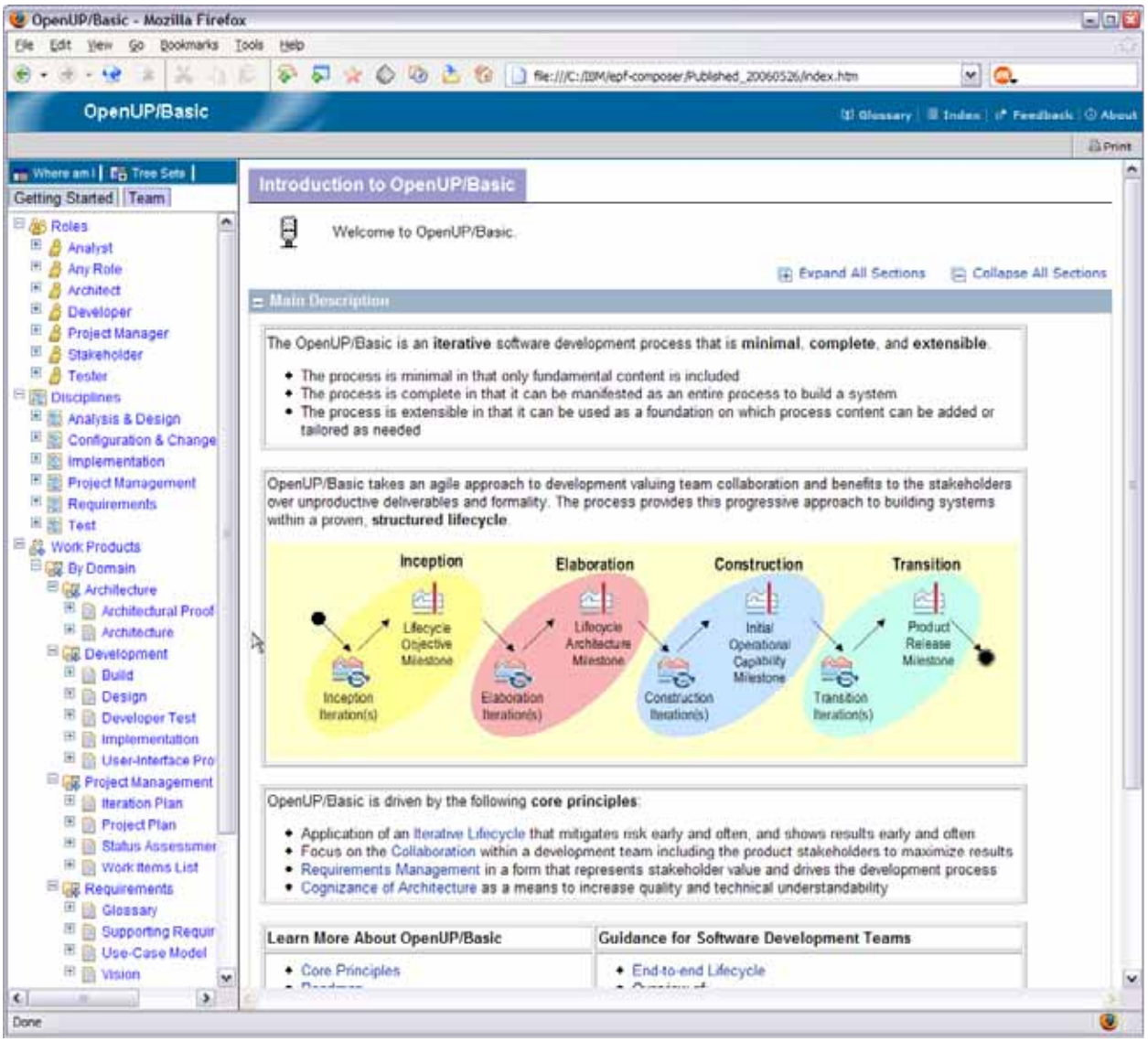


Figure C.7 - Overview to OpenUp/Basic showing its key roles, disciplines that categorize tasks, and work products as well as the overall lifecycle model

Figure C.7 provides an overview of how OpenUP/Basic is presented to the end-user. The underlying method content and process models for OpenUP/Basic have been created and are managed with EPF Composer, which is a SPEM 2.0 implementation, comprises of instances of all major SPEM 2.0 concepts.

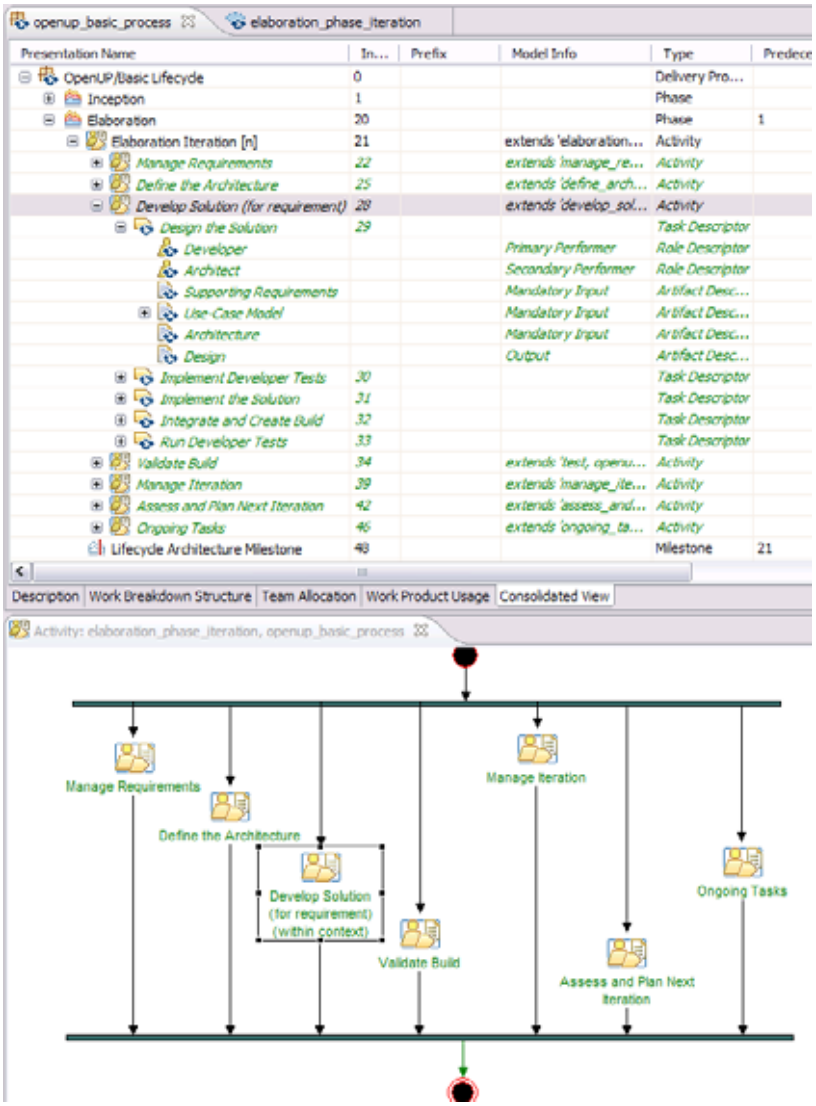


Figure C.8 - Typical OpenUP process presented as a breakdown structure as well as activity diagram

OpenUP provides a library of reusable method content elements such as roles, work products, and task definitions, as well as guidance. It also provides a list of Process Patterns for typical development situations that can be used to assemble larger organization and project specific Process Patterns and delivery processes. Figure C.8 drills into the OpenUP reference delivery process that represents an example of how OpenUP's Process Patterns can be assembled to an end-to-end process. The figure shows how the process is modeled using the SPEM 2.0 representation that allows it to be presented as an extended breakdown structure presenting performing roles and input/output work products as well as a UML 2 activity diagram.

OpenUP can be downloaded at http://www.eclipse.org/epf/.

C.4 MDA Process (OpenUP/MDD)

OpenUP/MDD (http://www.eclipse.org/epf/openup_component/mdd.php) is an extension to OpenUP/Basic (see C.2, 'Microsoft Solution Framework Agile Case Study') that models OMG's Model-Driven Architecture (MDA) approach. The OpenUP/MDD method content and process models represent the MDA approach with six key Role Definitions: Platform Expert (depicted in Figure C.9), Business Analyst, Domain Expert, Language Engineer, Test Designer, and the Transformation Specifier. These roles are responsible for altogether 57 different Work Products, and perform 90 standard Tasks.

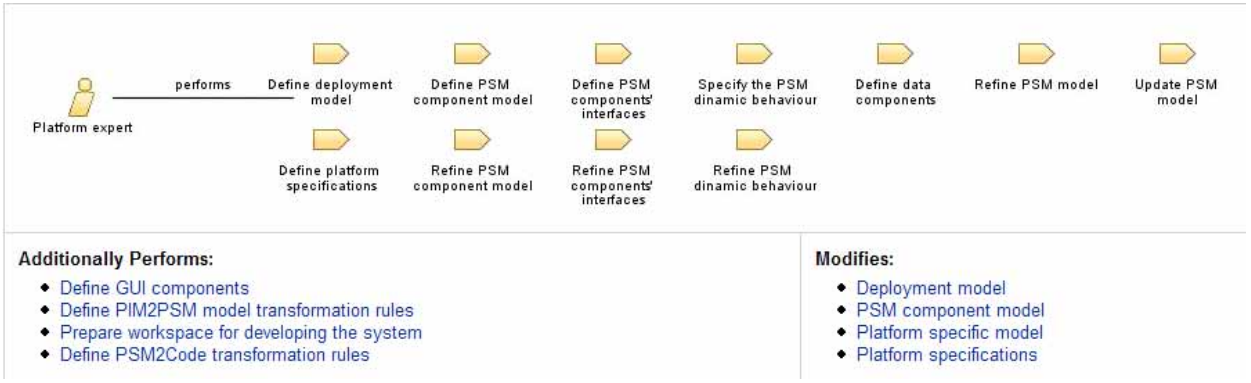


Figure C.9 - 'Platform Export' Role Overview

This method content has then been applied to 21 Process Patterns as depicted in Figure C.10 that describe the key processes to be performed in an MDA project.

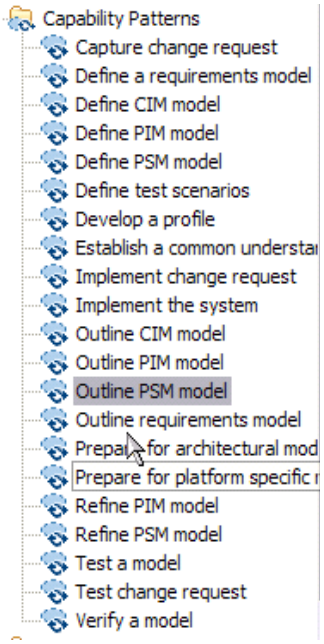


Figure C.10 - Process Patterns realizing MDA

Figure C.11 shows one of these Process Patterns in more detail depicting its two key presentations. It shows the pattern being presented as a breakdown structure (left) and as an UML 2 activity diagram (right).

SPEM 2.0 Specific Breakdown Structure

Presentation Name	Index	Model Info	Type	Predecessors
Outline PSM model	0		Capability Pattern	
Mark PIM model	1		Task Descriptor	
Develop a platform specific metamodel	2		Task Descriptor	1
Language engineer		Primary Performer	Role Descriptor	
Platform metamodel		Output	Artifact Descriptor	
Select a platform specific metamodel	3		Task Descriptor	1
Language engineer		Primary Performer	Role Descriptor	
Platform metamodel		Output	Artifact Descriptor	
Define PIM2PSM model transformation rules	4		Task Descriptor	2,3
Transformation specifier		Primary Performer	Role Descriptor	
System analyst		Secondary Performer	Role Descriptor	
Platform expert		Secondary Performer	Role Descriptor	
Platform metamodel		Mandatory Input	Artifact Descriptor	
Architecture metamodel		Mandatory Input	Artifact Descriptor	
PIM2PSM transformation rules		Output	Artifact Descriptor	
Generate a PSM from PIM	5		Task Descriptor	4
Platform metamodel		Mandatory Input	Artifact Descriptor	
Architecture model (PIM)		Mandatory Input	Deliverable Descriptor	
Platform specific model		Output	Deliverable Descriptor	

'Linked' UML 2.0 Activity Diagram

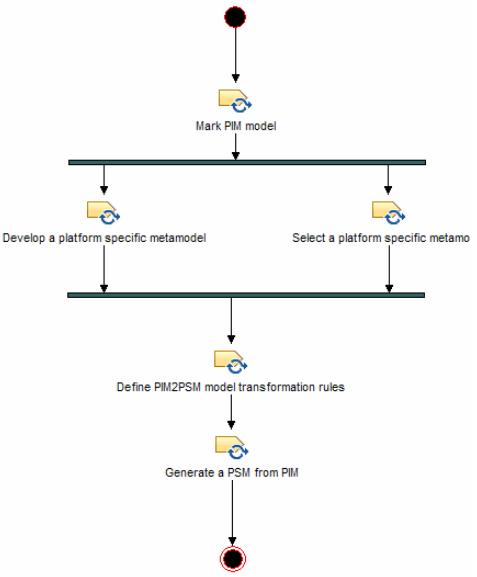


Figure C.11 - 'Outline PSM Model' Capability

Download and review OpenUP/MDD from here: http://www.eclipse.org/epf/openup_component/mdd.php.

C.5 Tivoli Unified Process (ITIL-based process)

The IBM Tivoli Unified Process (ITUP) provides detailed documentation of IT Service Management processes and governance based on industry best practices. The processes described within ITUP are strongly aligned with the ITIL (Information Technology Infrastructure Library). ITIL provides high-level guidance for what should be implemented, but does not provide guidance for how to implement. ITUP models detailed processes using SPEM 2.0 concepts and guidance to help users understand these processes and their relationships, making ITIL recommendations directly actionable utilizing SPEM's Process Pattern and process reuse capabilities (see Section 9.2, "Activity Use Kind," on page 48). ITUP also contains mappings to other process models, such as CobiT (Control Objectives for Information and related Technology).

Figure C.12 - Some of the 17 IT processes (left) represented by ITUP as Process Patterns and so-called process scenario that also have been assembled as Process Patterns using elements from the IT processes as building blocks

ITUP provides detailed method content models centered around 68 role definitions in IT Management documenting the artifacts they are responsible for, the tasks they perform, and detailed guidance including step by step description for using tools to support these tasks. This method content is applied in 17 core IT processes (see a selection in Figure C.12, left and an example for an IT process in Figure C.13) that represent the fundamental Process Patterns that can be used as a process pool to assemble organization specific processes. ITUP provides examples for such organization-specific processes, such as so-called process scenarios (Figure C.12, right and an example for a process scenario in Figure C.14) that are also Process Patterns that have been assembled with elements from the IT processes using activity use to link these elements.

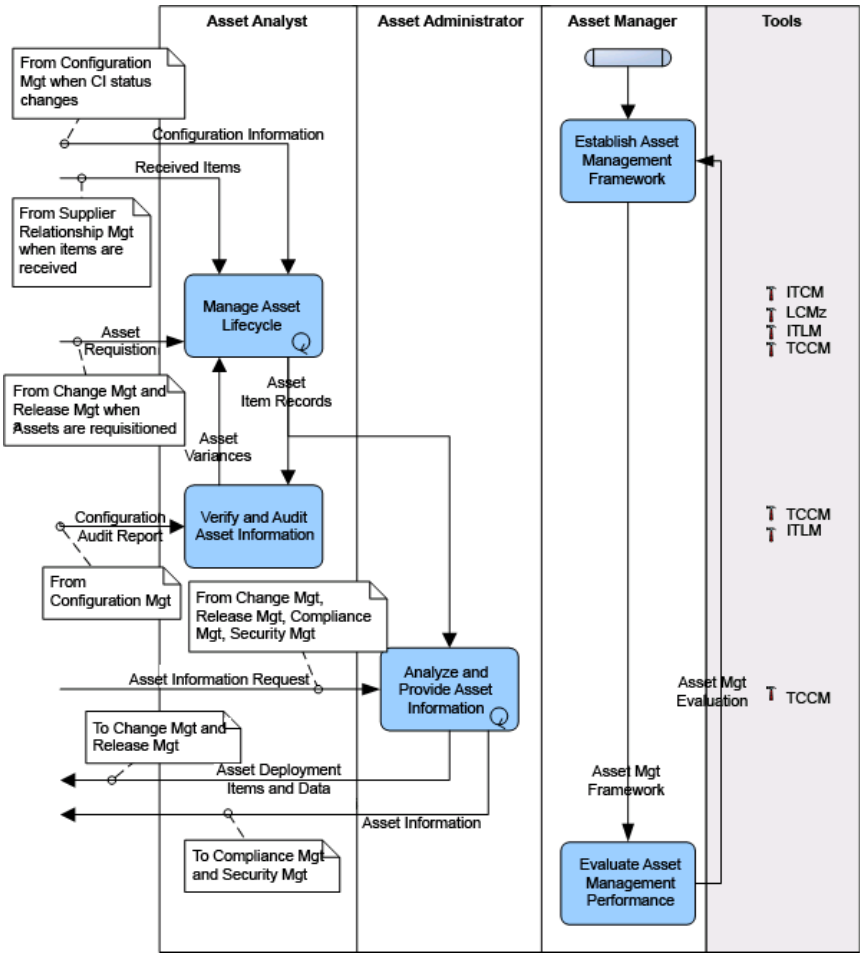


Figure C.13 - Asset Management IT Process represented as a SPEM 2.0 Process Pattern that uses ITUP's own proprietary behavior modeling approach to present the process' workflow

Figure C.13 depicts the ITUP IT process for Asset Management. This IT process has been modeled as a SPEM 2.0 Process Pattern using SPEM 2.0 activities. Furthermore, it as been modeled with ITUP's own a behavior model representation that is very similar to UML 2 activity diagrams, but provides some ITUP specific extensions. Each activity in Asset Management Process Pattern has been mapped to a node of Figure C.13 using the associations of linking SPEM 2.0 activities with external behavior model activities defined in Chapter 10 of this specification keeping a consistent link between the SPEM 2.0 model and the external behavior model.

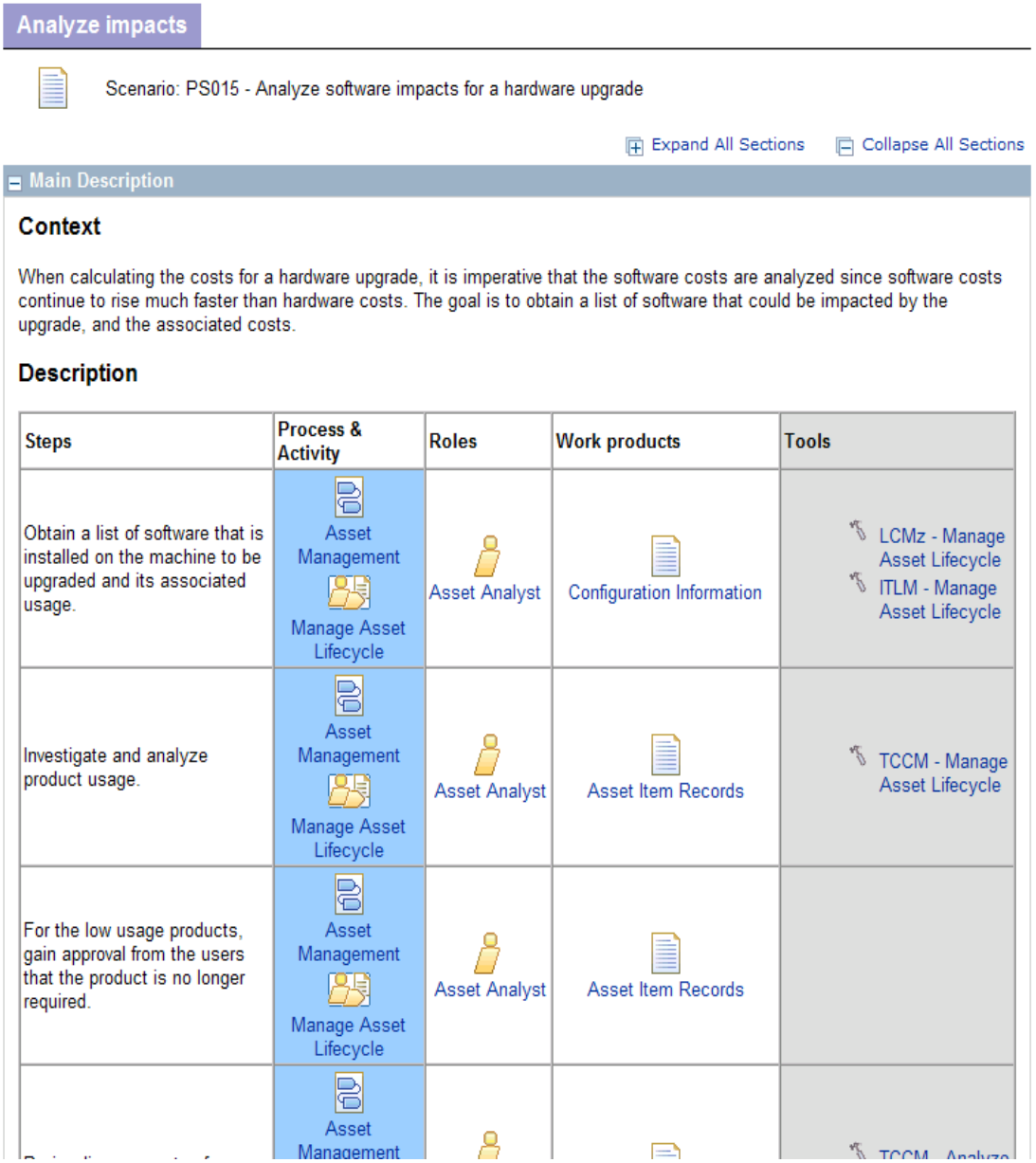


Figure C.14 - A process scenario illustrating how IT processes work together to perform common IT functions

Figure C.14 depicts an excerpt of ITUP's Analyze Impacts process scenario, which has been modeled as a SPEM 2.0 Process Pattern. This pattern has been assembled by directly reusing activities that have been defined in different IT Processes. For example, you see in Figure C.15 the Manage Asset Lifecycle activity from the upper left of Figure C.14. This activity reuse has been modeled using the activity use association between activities and the Extends activity use kind (see Section 9.2).

ITUP can be downloaded at http://www-306.ibm.com/software/tivoli/features/it-serv-mgmt/itup/overview.html.

C.6 PMBOK (Sierra System Process Interface Portals)

Sierra System used the SPEM 2.0 concepts to re-model the Project Management Body of Knowledge (PMBOK, IEEE Std 1490-2003). Their SPI – PM Portal models the nine PMBOK knowledge areas as method content and guidance as well as represents the five basic process groups as SPEM 2.0 Process Patterns as depicted in Figure C.15.

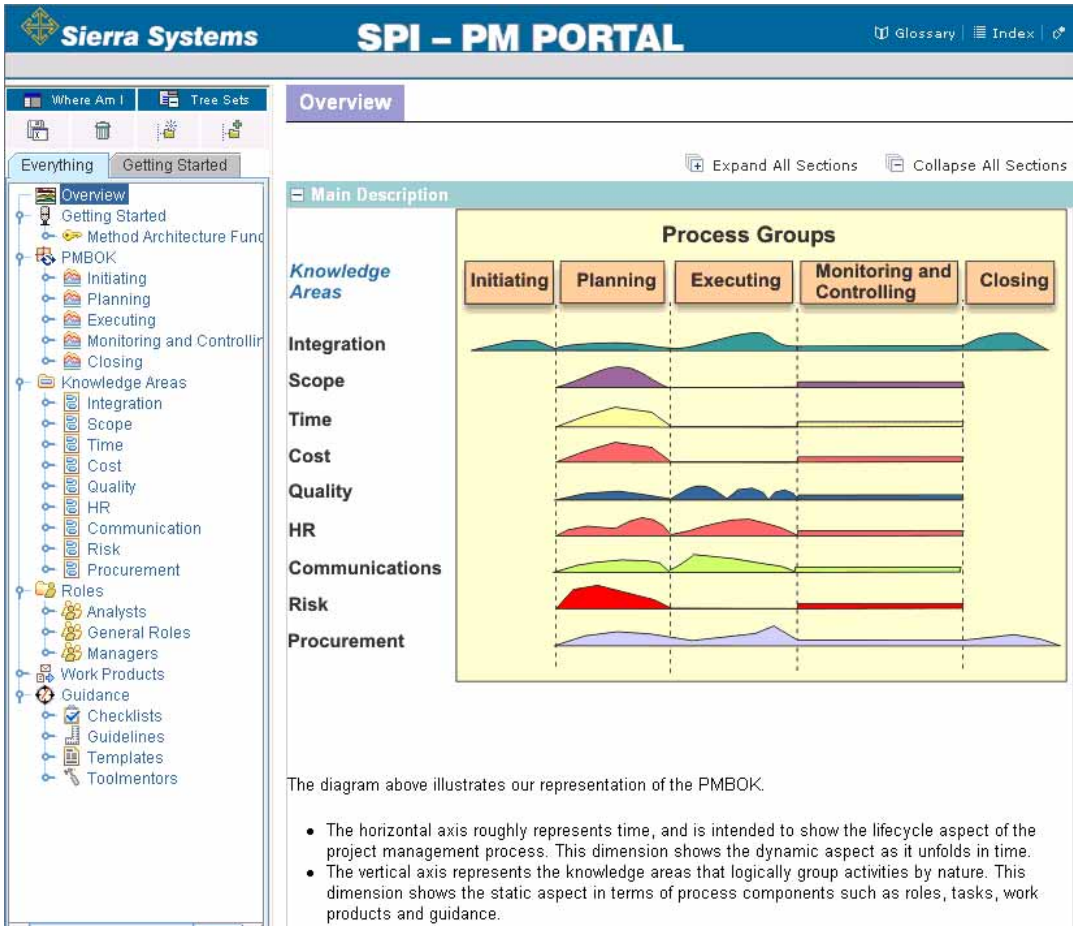


Figure C.15 - Overview to the SPI – PM Portal PMBOK representation

Sierra modeled PMBOK using all major SPEM 2.0 concepts. For example, it represents key roles such as the Project Manager depicted in Figure C.16, standard task definitions, and work products used and produced by these tasks.

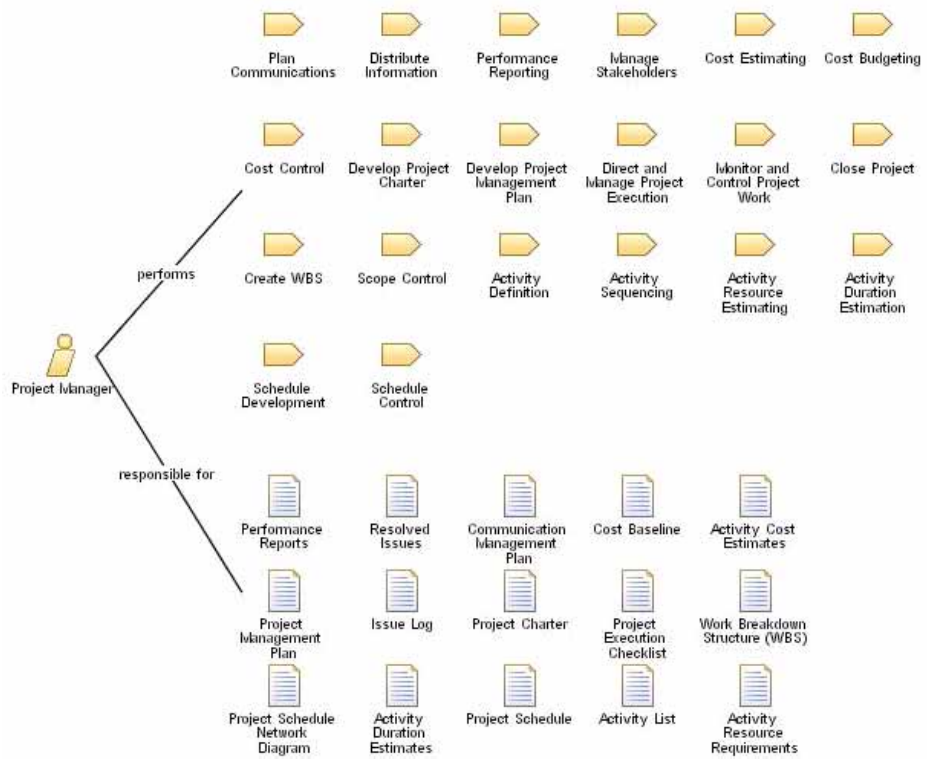


Figure C.16 - Overview to the Project Manager role

Sierra also models key guidance provided by PMBOK such as checklists for PMBOK work products and tasks such as Activity Definition, Cost Baseline, or Detailed Scope Statement for which each individual checkpoint has been modeled using SPEM's Section (11.6) concept. Other guidance provides links to the original PMBOK documentation or guides such as PMI's PMBOK Guide.

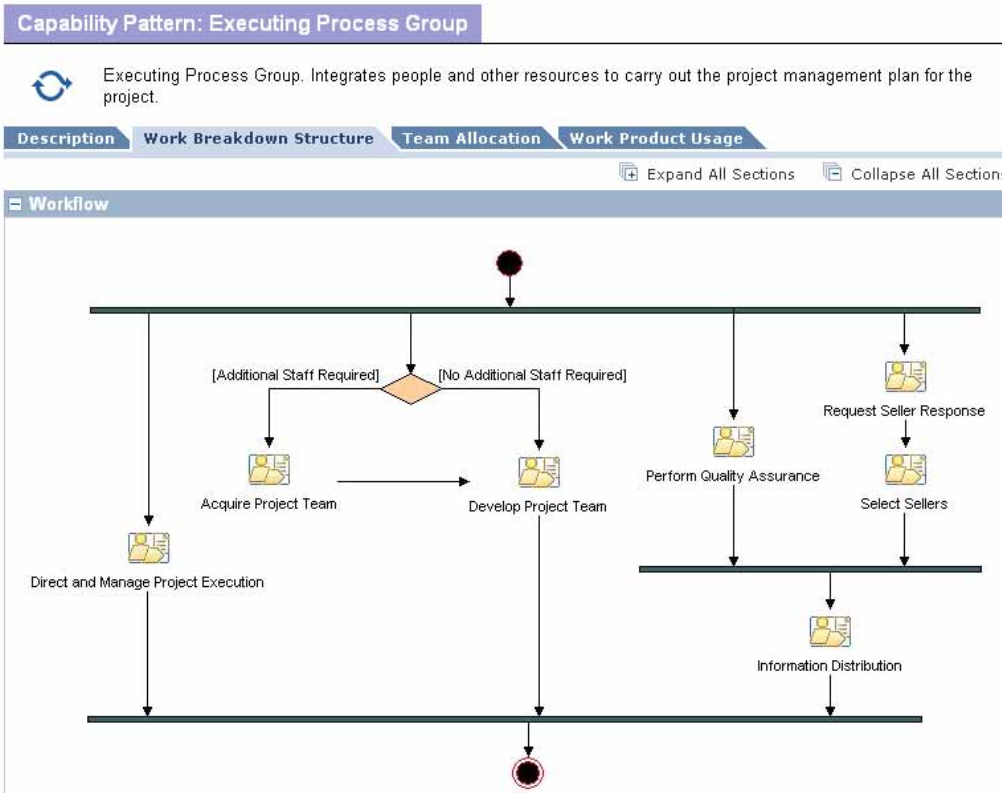


Figure C.17 - PMBOK Process Group modeled as an activity diagram

Finally, Figure C.17 shows an example of how a PMBOK process group is modeled as a SPEM 2.0 Process Pattern presented using an UML 2 Activity diagram.

C.7 SOA Governance Lifecycle and Management Method

In addition to the Software and IT management processes shown so far, the Software and Systems Process Engineering Meta-model SPEM 2.0 has also been used to represent organizational governance processes. The IBM SOA Governance Lifecycle and Management Method defines best practices and processes for establishing SOA governance in an enterprise. It defines processes for projects, structured along the IBM SOA Governance Lifecycle phases, to introduce or improve SOA Governance in an organization. This process does not perform SOA Governance, but enables an organization to exercise good governance through the identification, definition and deployment of governance roles, chains of responsibilities, processes and mechanisms for measurements, policy, and control.

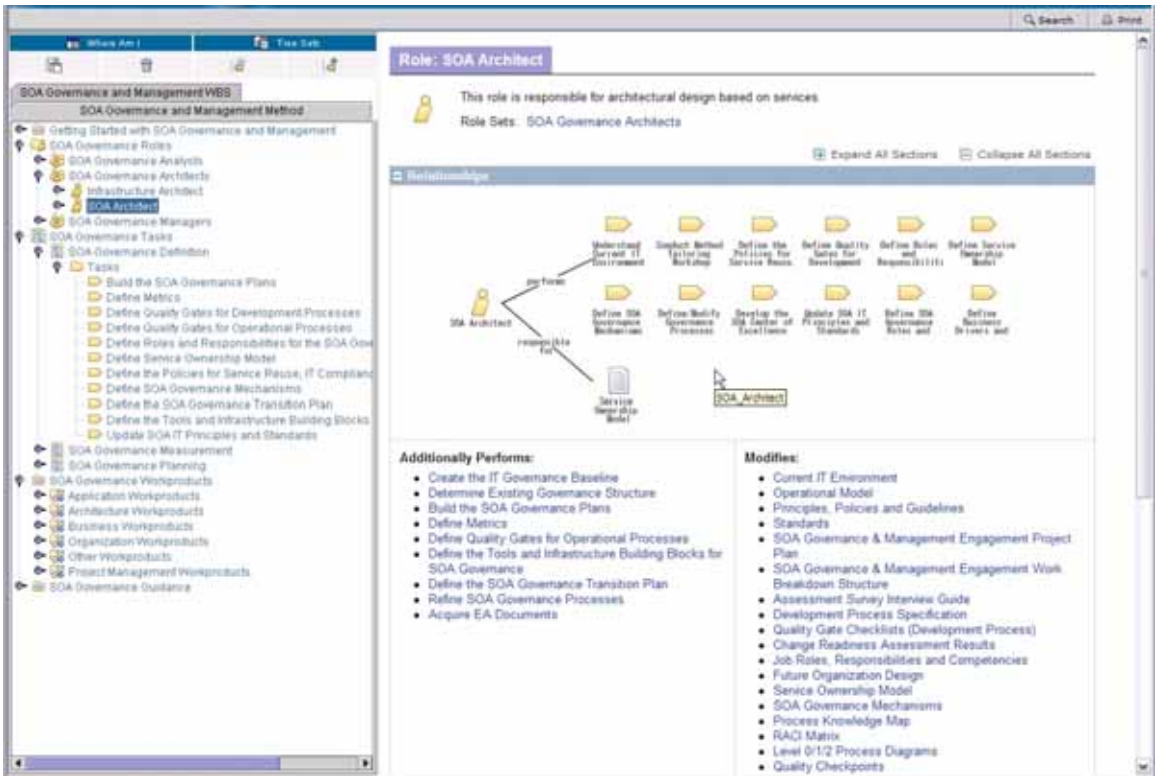


Figure C.18 - A role defined by the SOA Governance Method defining its key tasks, responsible work products, and work products the role modifies when performing the tasks

Figure C.18 depicts the key role of SOA Architect of the SOA Governance Lifecycle and Management Method. It shows on the right hand side that this role has been modeled as a SPEM 2.0 role definition performing a list of tasks and being responsible for a set of artifacts. The tree browser on the left hand side of Figure C.18 depicts a hierarchy of SPEM 2.0 Content Categories (see Section 11.1) that have been modeled to classify the role as a SOA Governance Architect. The tasks displayed have been categorized as well, in this case by a content category labeled SOA Governance Definition. The examples illustrate how SPEM 2.0 users can utilize the Custom Category concept to define hierarchical indexing and organization structures to manage and present large amounts of modeled method content.

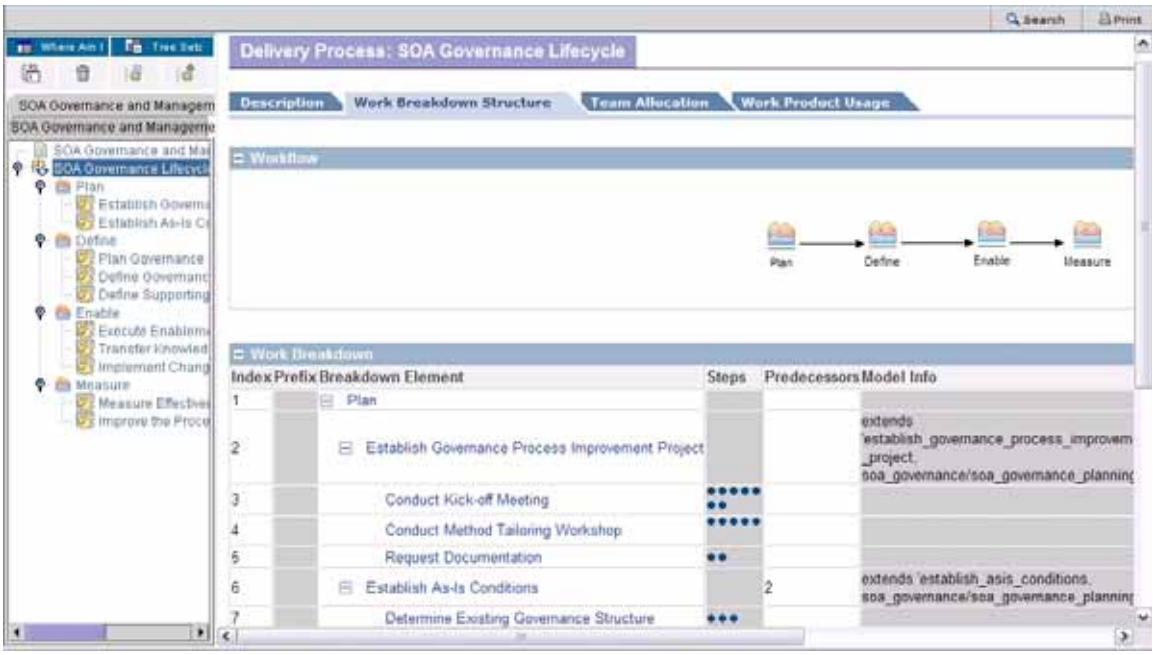


Figure C.19 - SOA lifecycle for introducing or improving SOA Governance in an organization

The method content shown in Figure C.18 has been applied to a four phase SOA lifecycle model represented in SPEM 2.0 as a delivery process as depicted in Figure C.19. The delivery process is presented in Figure C.19 as a UML 2 Activity diagram as well as a breakdown structure.

The SOA Governance Lifecycle and Management Method is available at: http://www.ibm.com/software/solutions/soa/gov

C.8 OnDemand Process Asset Library (CMM-based Application Services Process)

A key goal for the design of SPEM 2.0 was to provide effective process modeling and reuse mechanisms for organizations to build up or acquire libraries of reusable method content and process building blocks. Such libraries can represent repositories of best software or system development practices and capabilities that can be effectively utilized by these organizations to build up their group or project-specific processes. Such process repositories could be repopulated with practice and capability building blocks defined by standards or de-facto standards such as CMMi. Such building blocks could then be selected and assembled into a process based on the maturity level and the process needs of organizations and development teams.

IBM's OPAL (OnDemand Process Asset Library) represents such a CMMi process repository developed with SPEM 2.0. It has been developed by a CMM-level 5 organization within the IBM Global Services Application Services. It governs the way Application Services staffs, plans, and executes projects and programs, as well as performs related organizational functions. It has been designed as single reference point for all teams regarding project management, program management, and related engineering.

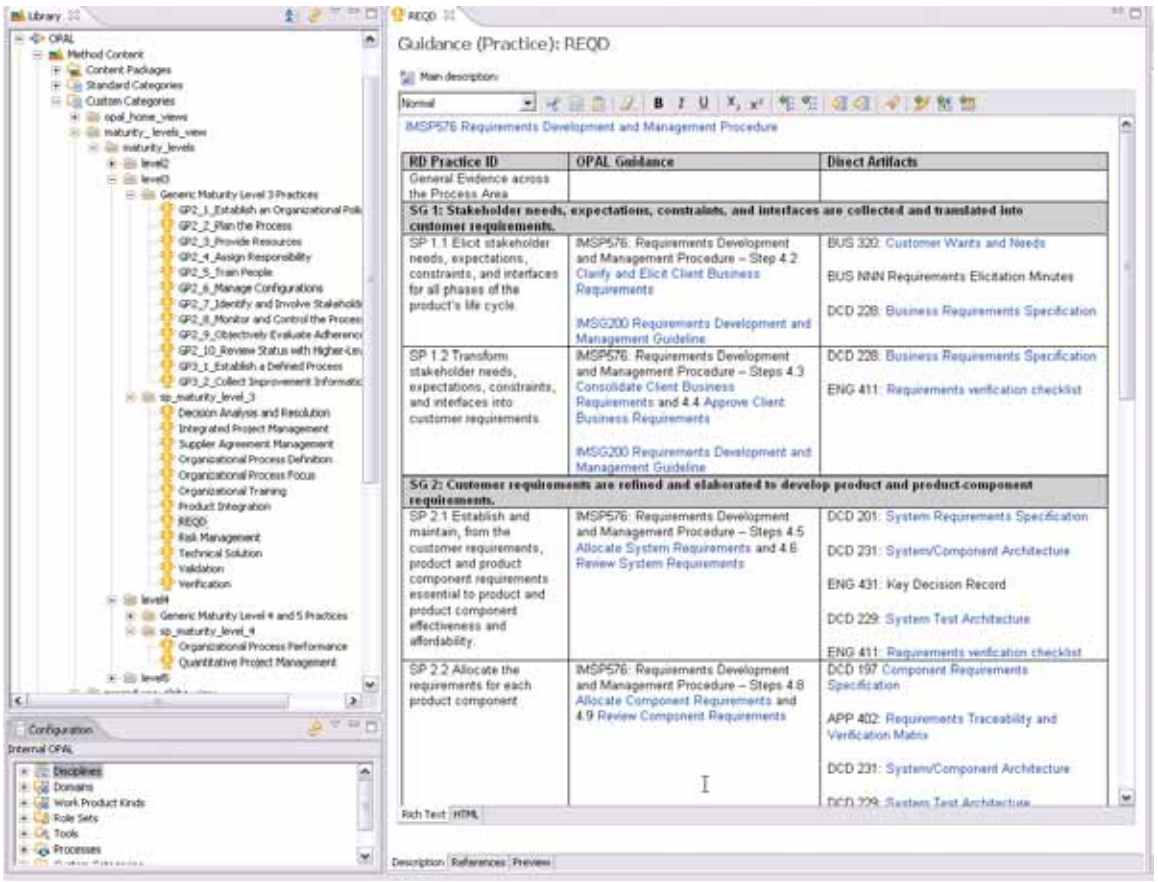


Figure C.20 - Documented CMM Practices are linked to enactable process models (Process Patterns) and artifacts used and produced in the practice

Using SPEM 2.0 Content Categories and different Guidance Kinds (see Section 18.3, “Guidance Kinds,” on page 162), the OPAL process asset library defines, organizes, and documents several views on the detailed method content and the processes it contains. Among others, it lets process users access content via CMM maturity levels. As depicted in Figure C.20, for each maturity level, key CMM-practices are documented and then mapped to key artifacts. These practices are also mapped to so-called OPAL Procedures that define mini-processes that realize the documented practices that have been modeled as SPEM 2.0 Process Patterns.

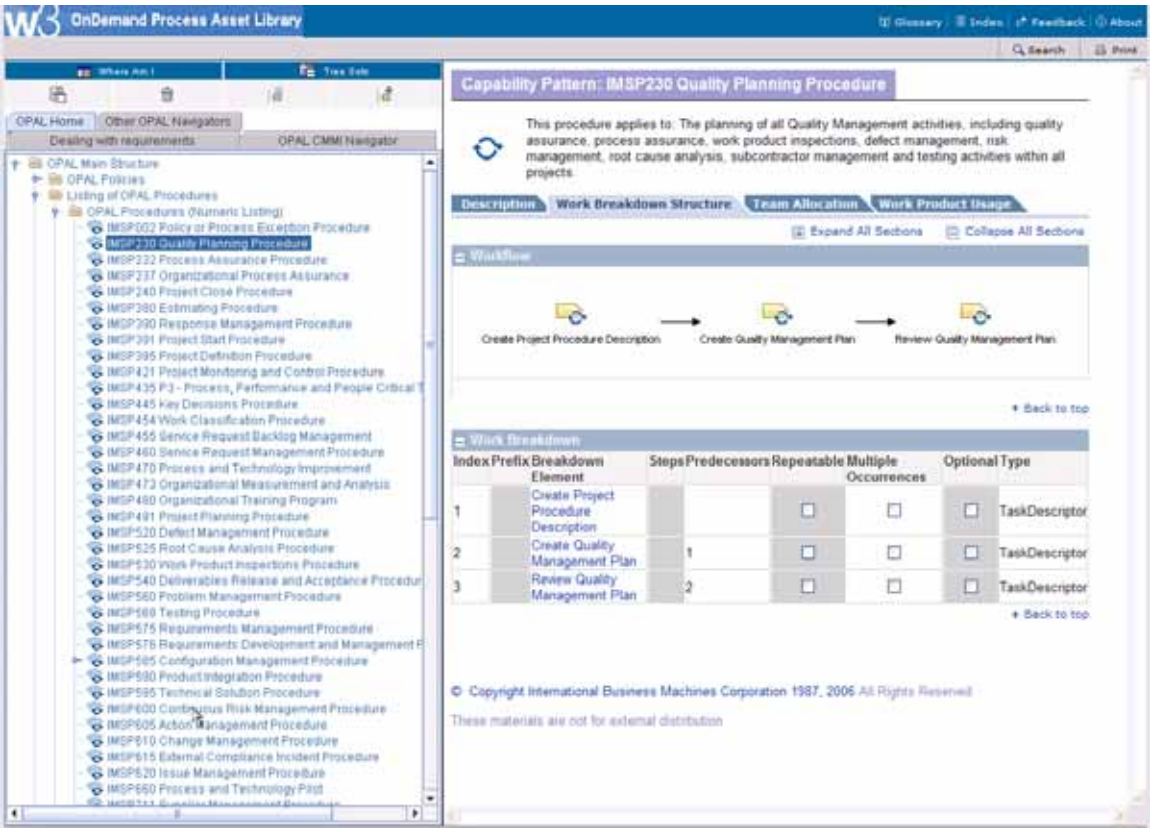


Figure C.21 - List of Process Patterns that model so-called OPAL procedures that can be used as reusable building blocks to assemble larger end-to-end processes

Figure C.21 depicts a list of Process Patterns (referred to as Capability Patterns) that have been modeled as SPEM 2.0 Process Patterns in OPAL. These patterns represent building blocks for assembling processes. OPAL users can start selecting patterns by reviewing the CMM Practices depicted in Figure C.20 and then navigating to these patterns that show how the practice can be realized. The Process Pattern can then be applied to a delivery process using SPEM 2.0 concepts such as the activity use kind Extends, and then finally tailored towards specific needs using SPEM 2.0 activity use kinds local contributions and local replacement.

C.9 E&TS Application Specific Integrated Circuits Method

This case study shows that SPEM 2.0 has not only been used to model software development processes, but processes for developing hardware systems as well. Engineering & Technology Services (E&TS) is a division of IBM focused on design and implementation of electronic components, systems, and consulting for external customers spanning a wide variety of industries from aerospace to consumer to semiconductor. SPEM 2.0-based methods were recently developed to improve project consistency and provide documentation for training and business growth. They are used by both project managers to better plan and guide execution, as well as the technical teams to document their development approach.

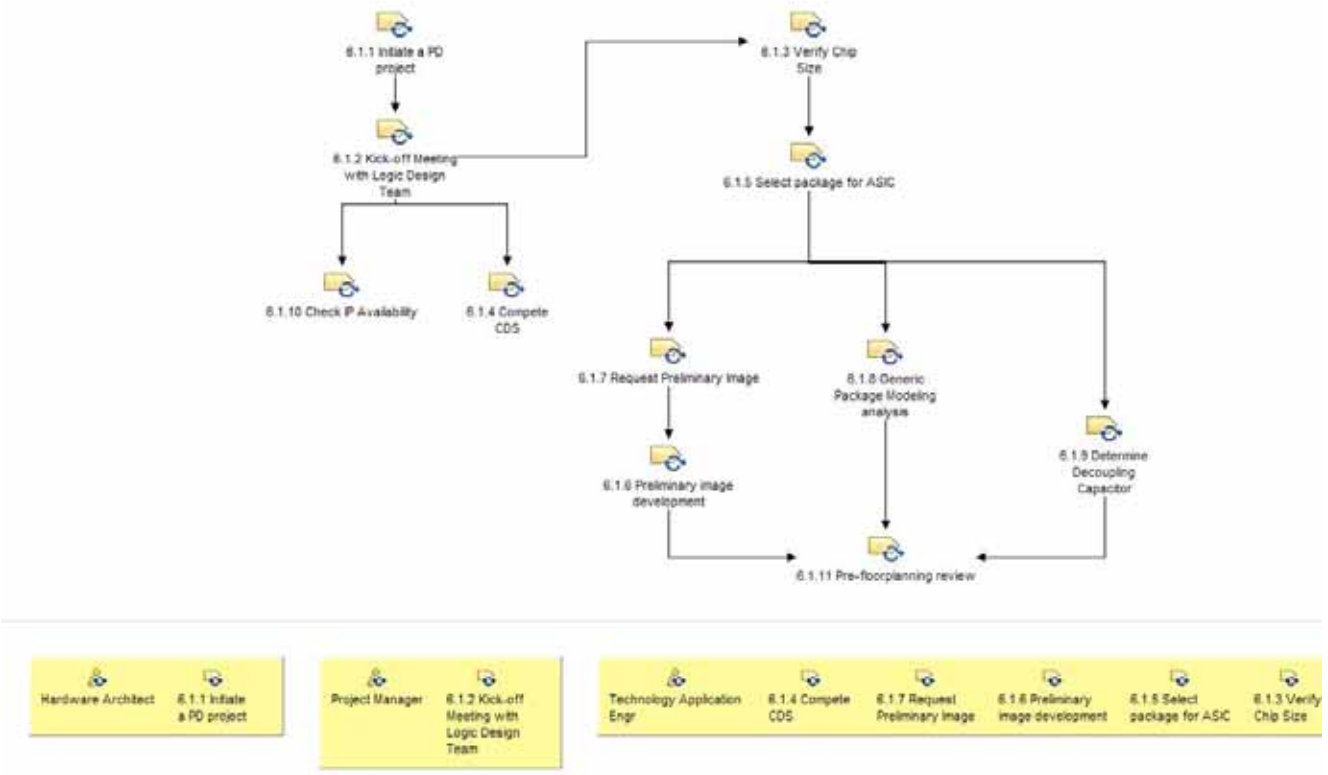


Figure C.22 - A workflow in the ASIC method (top) defining control flow among task uses as well as list of the same task uses grouped by performing role

Application Specific Integrated Circuits (ASIC) are electronic components which are designed and built to solve a particular need. The first three phases of the E&TS ASIC Method focus on the design and specification of the hardware solution, followed by preparing the design for manufacture, semiconductor fabrication of the design and finally, validating that the hardware component operates the way in which it was designed. Figure C.22 shows one of workflows defined in E&TS ASIC method's Design phase called Physical Design. This workflow represents a SPEM 2.0 activity that contains task uses as well as role uses performing these tasks.

C.10 SPW-6 Software Process Example

The ISPW-6 Software Process Example was originally published in the "Proceedings of the 6th International Software Process Workshop: Support for the Software Process" (Hakodate, Hokkaido, Japan; 28-31 October, 1990; edited by Takuya Katayama; published by IEEE Computer Society Press, 1991), under the title "Software Process Modeling Example Problem." It has been designed to aid in understanding and comparing various approaches to software process modeling. Members of the Laboratoire d'Informatique Paris 6 (LIP6) have modeled this example using SPEM 2.0 concepts. The full example is available on request from Reda Bendraou, Reda.Bendraou@lip6.fr.

Back to top

Work Breakdown

Index	Prefix	Breakdown Element	Steps	Predecessors	Model Info	Repeatable	Multiple Occurrences	Optional	Planned	Type
1		Dev_Change_and_Test_Unit				✔			✔	Iteration
2		Schedule_Assign				✔			✔	Activity
3		Schedule_and_Assign_Tasks	○○○			✔				Task Descriptor
4		Modify_and_Review_Design		2					✔	Activity
5		Modify_Review Design Iteration			extends 'Modify_Review Design Iteration, ISPW-6_DevChange_and_TestUnit'	✔			✔	Activity
6		Modify_Design	○○			✔				Task Descriptor
7		Review_Design	○○○○	6		✔				Task Descriptor
8		Modify_Code		4		✔			✔	Activity
9		modify_code	○○							Task Descriptor
10		Tests		2					✔	Activity
11		Tests_Iteration			extends 'Tests_Iteration, ISPW-6_DevChange_and_TestUnit'				✔	Activity
12		Modify_Test_Plans	○			✔				Task Descriptor
13		Modify_unit_test_Package	○	12						Task Descriptor
14		Test_Unit	○○	13						Task Descriptor
15		MonitorProgress		2					✔	Activity
16		monitor_progress	○○							Task Descriptor

Figure C.23 - General ISPW-6 example breakdown structure

The ISPW-6 example is scoped as a relatively confined portion of the software change process. It focuses on the designing, coding, unit testing, and management of a rather localized change to a software system. This is prompted by a change in requirements, and can be thought of as occurring either late in the development phase or during the support (maintenance and enhancements) phase of the life-cycle. The example is presented as a breakdown structure describing activities and tasks, which LIP6 represented as a SPEM 2.0 process as depicted in Figure C.23.

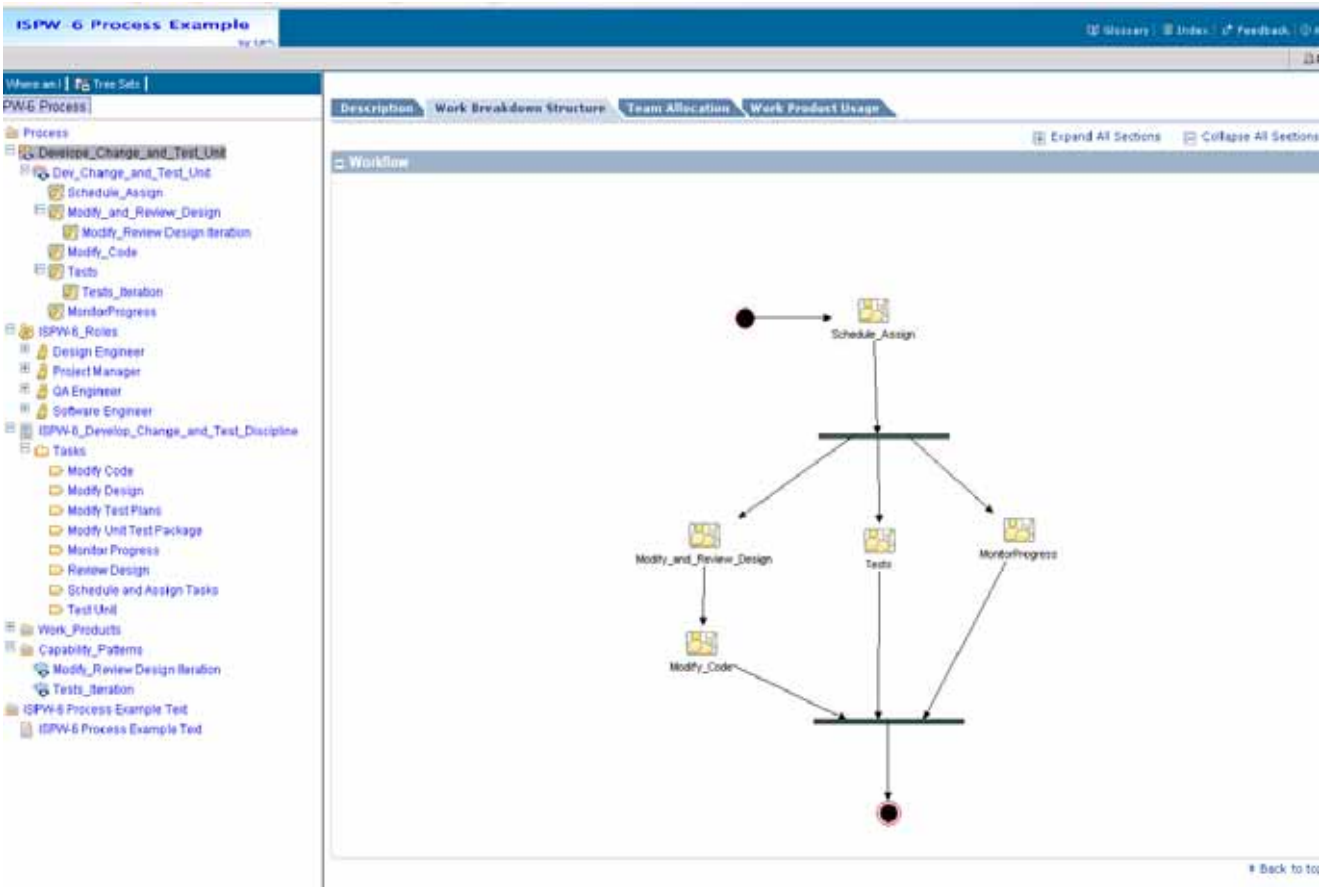


Figure C.24 - Flow of activities in the ISPW-6 example

The flow of key activities of this breakdown structure has been represented as the UML 2 activity diagram of Figure C.24.

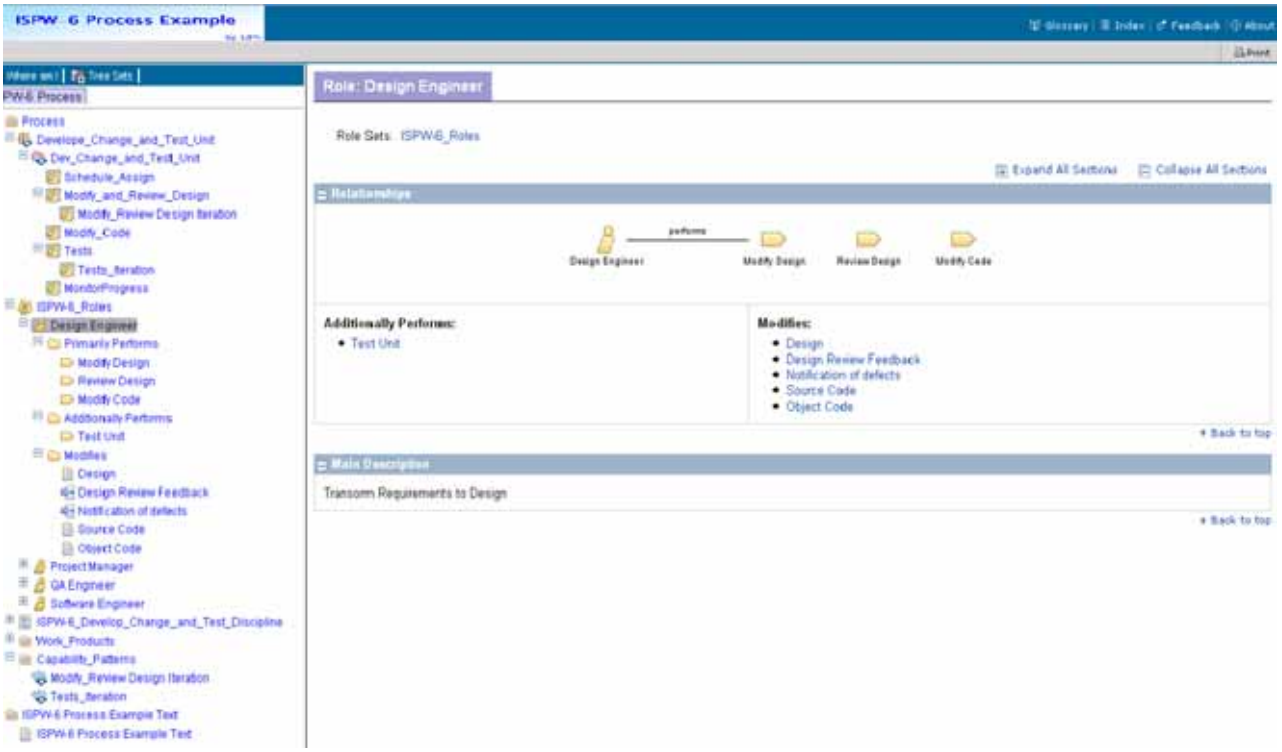


Figure C.25 - Role Definition for Design Engineer in the ISPW-6 Example

Finally, LIP6 extracted method content from the breakdown structure representation and modeled this as role, work product, and task definitions. Figure C.25 depicts the role definition for Design Engineer and the task this role performs in the example.

C.11 Money-Lover (Process for Investment Clubs)

The final case study shows that SPEM 2.0 concepts can also be used to represent and document processes that are not related to software and systems development projects at all. The web site http://www.money-lover.us publishes best practices and processes for investing and investment clubs based on the BetterInvesting (NAIC) organization's recommendations. It uses SPEM 2.0 concepts to represent and document these models.

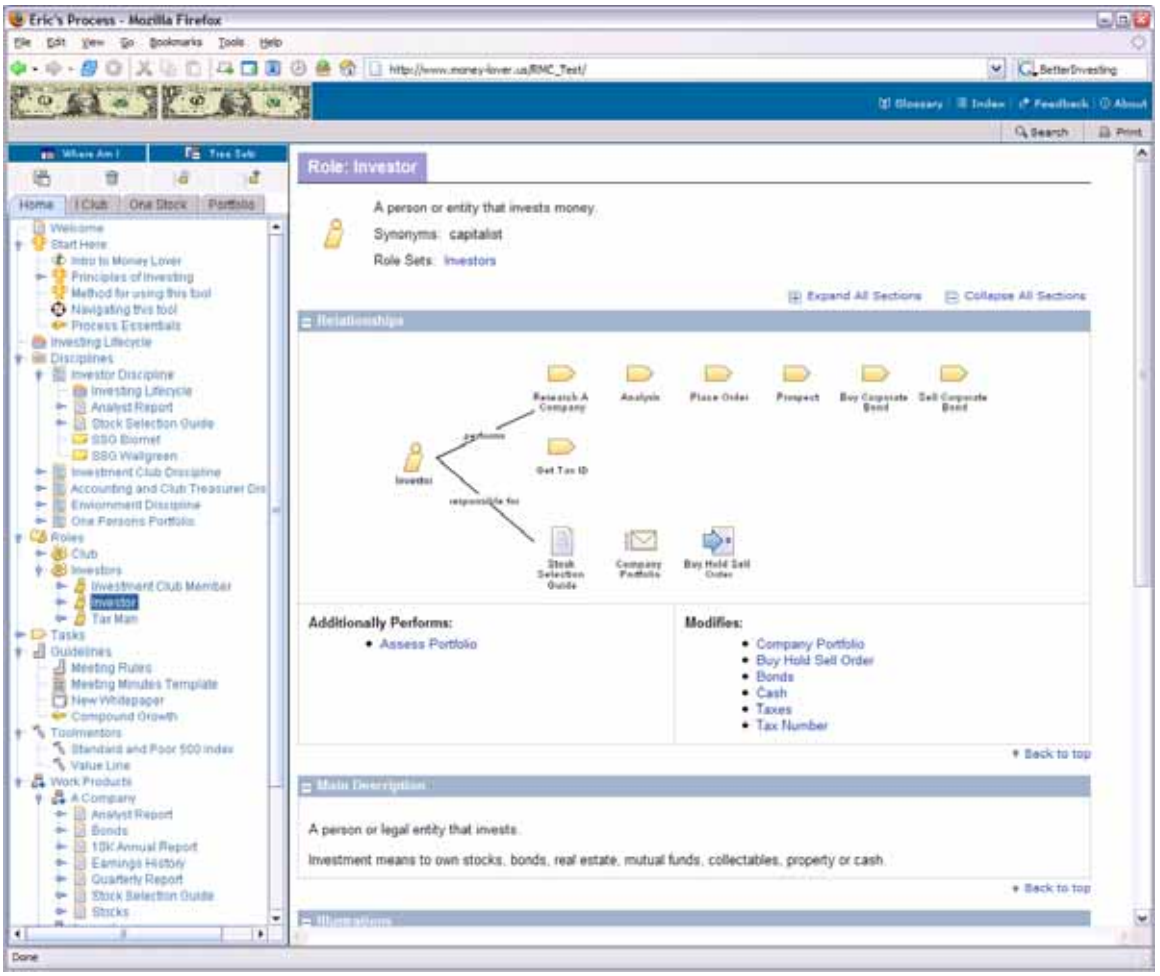


Figure C.26 - The Money Lover website

Figure C.26 shows a glimpse at the SPEM 2.0 Content Categories hierarchy Money Lover created to organize its method content, which is also used navigate the web site. On the right-hand side, it depicts the model of the role definition Investor with the work products it is responsible for (or better in this case it cares about). It also models tasks definitions that define the Investor as a performer. These task document the 'things' an investor does providing a role-based information scheme of presenting and retrieving instructions and tips for better investing.

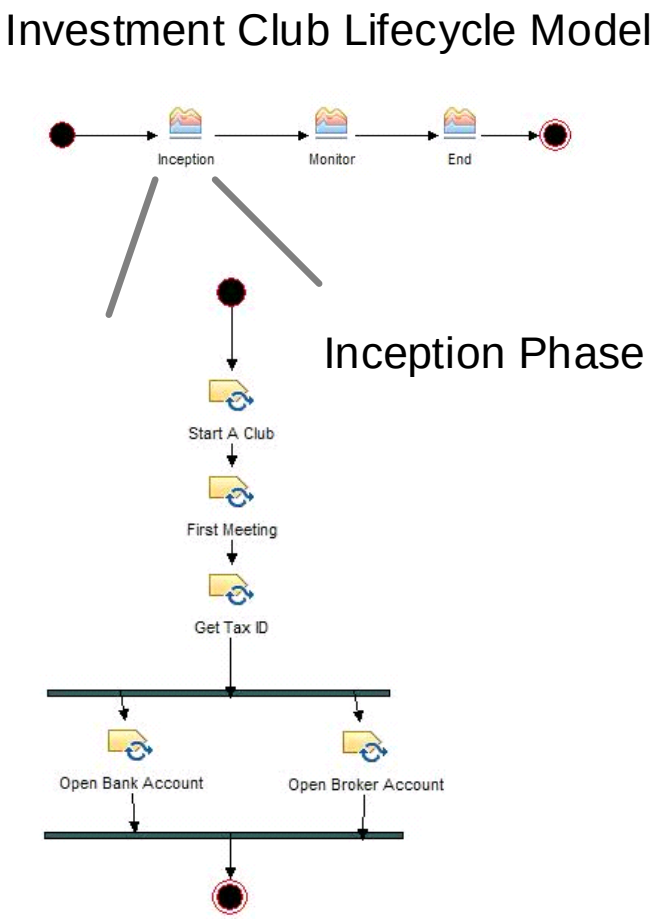


Figure C.27 - Investment club lifecycle model defining three phases and the detailed workflow for the first phase

In addition to modeling individual investment practices as method content, Money Lover also defines a set of simple processes that define workflows for investment activities that go through a 'lifecycle.' For examples, it defines a Portfolio lifecycle or an Investment Club lifecycle as depicted in Figure C.27.

INDEX

A
Activity 46, 97, 119
Activity Kinds 155
Activity Use 49
Artifact 166

B
Background 8
Breakdown Element 99
business flow 147

C
Capabilities 11
Case studies and examples 187
Category 74
Category Kinds 160
Checklist 163
Commercial product 22
Compliance points 1
Composite Role 100
Concept 163
Content Description 76
Contribution Activity 52
Core meta-model package 3, 35

D
Default Responsibility Assignment 83
Default Task Definition Parameter 84
Default Task Definition Performer 85
Deliverable 166
Delivery Process 157
Describable Element 77
Discipline 161
Domain 161

E
enactment machine 149
Estimate 163
Example 163
execution language 147
Extensible Element 36

G
Guidance 78
Guidance kinds 162
Guideline 163

I
Icon presentation 26
Introduction 9
issues/problems xii
Iteration 156

K
Kind 37

L
Library Configuration 118
Local contribution 52
Local replacement 52

M
Managed Content package 3, 73
Mapping SPEM 1.1 to SPEM 2.0 151
Meta-model packages 2
Method Configuration 120
Method Content Element 85
Method Content Kind 101
Method Content Package 3, 81, 102
Method Content Packageable Element 103
Method Content Use 104
Method Library 123
Method Library Element 123
Method Library Packageable Element 123
Method Plugin 117, 124
Method plug-in package 4
Method Plugin Packageable Element 126
Metric 79
MOF 2.0-based Meta-Model 25

O
Object Management Group, Inc. (OMG) xi
OMG specifications xi
Optionality Kind 86
Outcome 165
Overview 25

P
Packages 3
Parameter direction kind 37
Phase 156
Planning Data 106
Practice 164
Process 156
Process Behavior package 3, 69
Process Component 127
Process Component Use 131
Process Kind 107
Process Package 108
Process Packageable Element 109
Process Pattern 158
Process Performer 57
Process Planning Template 160
Process Responsibility Assignment 58
Process Structure meta-model package 43
Process structure package 3
Process with methods package 4
Project Plans 147

Q
Qualification 86

R
References 7
Report 164
Reusable Asset 164
Roadmap 164
Role Definition 87
Role Set 161
Role Use 59, 110

S
Section 79, 132
SPEM 1.1 27
SPEM 2.0 Base Plug-in 155
SPEM 2.0 breakdown structures 148
SPEM 2.0 UML 2 Profile 1
SPEM Complete 4
State_ext 71
Step 88
superclasses 96
Supporting Materials 164

T
Task Definition 89
Task Definition Parameter 84
Task Definition Performer 85
Task Definitions 95
Task Use 112
Task uses with actions 29
Team Profile 113
Template 164
Term Definitions 164
Tool Category 161
Tool Definition 91
Tool Mentor 165
Transition_ext 71
typographical conventions xii

U
UML 2 diagram interchange 1
UML 2 Infrastructure library 1
UML 2 stereotypes 169
UML Activities 28

V
Variability Element 133
Variability Examples 175
Variability Type 137

W
Whitepapers 165
Work Breakdown Element 60, 148
Work Definition 38
Work Definition Parameter 40, 71
Work Definition Performer 41
Work Definitions 28
Work Product Definition 92
Work Product Definition Relationship 93
Work Product Kinds 165
Work Product Port 142
Work Product Port Connector 142
Work Product Relationship Kinds 167
Work Product Use 62, 116
Work Product Use Relationship 63
Work Sequence 67
Work Sequence Kind 68